

## Contents

1 Basic	1
1.1 .vimrc	1
1.2 PBDS	1
1.3 pragma	1
1.4 Default Code	1
2 Graph	1
2.1 2SAT/SCC	1
2.2 BCC Vertex	2
2.3 BCC Edge	2
2.4 MinimumMeanCycle	2
2.5 MaximumCliqueDyn	2
2.6 DMST(slow)	3
2.7 DMST	3
2.8 VizingTheorem	3
2.9 MinimumCliqueCover	4
2.10CountMaximalClique	4
2.11Theorems	4
3 Flow-Matching	4
3.1 HopcroftKarp	4
3.2 KM	5
3.3 MCMF	5
3.4 GeneralGraphMatching	6
3.5 MaxWeightMaching	6
3.6 GlobalMinCut	7
3.7 BoundedFlow(Dinic)	8
3.8 GomoryHuTree	8
3.9 MinCostCirculation	8
3.10FlowModelsBuilding	9
4 Data Struture	9
4.1 LCT	9
4.2 Treap	10
4.3 CentroidDecomposition	10
4.4 HeavylightDecomposition	11
5 String	11
5.1 KMP	11
5.2 Z	11
5.3 Manacher	11
5.4 SuffixArray	11
5.5 SAIS	12
5.6 ACAutomaton	12
5.7 MinRotation	13
5.8 ExtSAM	13
5.9 PalindromeTree	13
6 Number Theory	13
6.1 Primes	13
6.2 ExtGCD	13
6.3 FloorCeil	14
6.4 FloorSum	14
6.5 MillerRabin	14
6.6 PollardRho	14
6.7 Fraction	14
6.8 ChineseRemainder	14
6.9 FactorialMod $p^k$	14

6.10QuadraticResidue	14
6.11MeisselLehmer	15
6.12DiscreteLog	15
6.13Theorems	15
6.14Estimation	16
6.15EuclideanAlgorithms	16
6.16Numbers	16
6.17GeneratingFunctions	16
7 Linear Algebra	16
7.1 GuassianElimination	16
7.2 BerlekampMassey	16
7.3 Simplex	17
8 Polynomials	17
8.1 NTT (FFT)	17
8.2 FHT	18
8.3 PolynomialOperations	18
8.4 NewtonMethod+MiscGF	18
9 Geometry	18
9.1 Basic	18
9.2 ConvexHull	19
9.3 SortByAngle	19
9.4 DisPointSegment	19
9.5 PointInCircle	19
9.6 PointInConvex	19
9.7 PointTangentConvex	19
9.8 CircTangentCirc	20
9.9 LineCircleIntersect	20
9.10LineConvexIntersect	20
9.11CircIntersectCirc	20
9.12PolyIntersectCirc	20
9.13MinkowskiSum	20
9.14MinMaxEnclosingRect	20
9.15MinEnclosingCircle	21
9.16CircleCover	21
9.17LineCmp	21
9.18Trapezoidalization	21
9.19TriangleHearts	22
9.20HalfPlaneIntersect	22
9.21RotatingSweepLine	23
9.22DeLaunayTriangulation	23
9.23VoronoiDiagram	24
10 Misc	24
10.1MoAlgoWithModify	24
10.2MoAlgoOnTree	24
10.3MoAlgoAdvanced	24
10.4HilbertCurve	24
10.5SternBrocotTree	24
10.6AllLCS	25
10.7SimulatedAnnealing	25
10.8SMAWK	25
10.9Python	25
10.10BineContainer	25

## 1.3 pragma

```
#pragma GCC optimize("Ofast,unroll-loops")
#pragma GCC target("avx,avx2,sse,sse2
,sse3,ssse3,sse4,popcnt,abm,mmx,fma,tune=native")
// chrono
::steady_clock::now().time_since_epoch().count())
```

## 1.4 Default Code

```
#include <bits/stdc++.h>
#define pii pair<int, int>
#define pll pair<ll, ll>
#define pdd pair<double, double>
#define ll long long
#define ld long double
#define i128 __int128
#define F first
#define S second
#define all(x) x.begin(), x.end()
using namespace std;

void solve() {}

int main() {
    ios::sync_with_stdio(0);
    cin.tie(0);
    solve();
}
```

## 2 Graph

### 2.1 2SAT/SCC

```
struct SAT { // 0-base
    int low[N], dfn[N], bln[N], n, Time, nScc;
    bool instack[N], istrue[N];
    stack<int> st;
    vector<int> G[N], SCC[N];
    void init(int _n) {
        n = _n; // assert(n * 2 <= N);
        for (int i = 0; i < n + n; ++i) G[i].clear();
    }
    void add_edge(int a, int b) { G[a].emplace_back(b); }
    int rv(int a) {
        if (a >= n) return a - n;
        return a + n;
    }
    void add_clause(int a, int b) {
        add_edge(rv(a), b), add_edge(rv(b), a);
    }
    void dfs(int u) {
        dfn[u] = low[u] = ++Time;
        instack[u] = 1, st.push(u);
        for (int i : G[u])
            if (!dfn[i])
                dfs(i), low[u] = min(low[i], low[u]);
            else if (instack[i] && dfn[i] < dfn[u])
                low[u] = min(low[u], dfn[i]);
        if (low[u] == dfn[u]) {
            int tmp;
            do {
                tmp = st.top(), st.pop();
                instack[tmp] = 0, bln[tmp] = nScc;
            } while (tmp != u);
            ++nScc;
        }
    }
    bool solve() {
        Time = nScc = 0;
        for (int i = 0; i < n + n; ++i)
            SCC[i].clear(), low[i] = dfn[i] = bln[i] = 0;
        for (int i = 0; i < n + n; ++i)
            if (!dfn[i]) dfs(i);
        for (int i = 0; i < n + n; ++i) SCC[bln[i]].emplace_back(i);
        for (int i = 0; i < n; ++i) {
            if (bln[i] == bln[i + n]) return false;
        }
    }
};
```

## 1 Basic

### 1.1 .vimrc

```
set ru nu cin cul sc so=3 ts=4 sw=4 bs=2 ls=2 mouse=a
ino {<CR> {<CR>}<C-o>0
ino jj <esc>
ino jk <esc>
map <F7> :w<CR>:!g++
    "% -std=c++17 -Wall -Wextra -Wshadow -Wconversion
    -fsanitize=address,undefined -g && ./a.out<CR>
ca Hash w !cpp -dD -P -fpreprocessed
\| tr -d "[:space:]" \| md5sum \| cut -c-6
```

### 1.2 PBDS

```
#include <bits/extc++.h>
using namespace __gnu_pbds;

#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
tree<int, null_type, less<int>, rb_tree_tag
, tree_order_statistics_node_update> bst;
// order_of_key(n): # of elements <= n
// find_by_order(n): 0-indexed

#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/priority_queue.hpp>
__gnu_pbds::priority_queue
<int, greater<int>, thin_heap_tag> pq;
```

```

    istrue[i] = bln[i] < bln[i + n];
    istrue[i + n] = !istrue[i];
}
return true;
}
};

```

## 2.2 BCC Vertex

```

int n, m, dfn[N], low[N], is_cut[N], nbcc = 0, t = 0;
vector<int> g[N], bcc[N], G[2 * N];
stack<int> st;
void tarjan(int p, int lp) {
    dfn[p] = low[p] = ++t;
    st.push(p);
    for (auto i : g[p]) {
        if (!dfn[i]) {
            tarjan(i, p);
            low[p] = min(low[p], low[i]);
            if (dfn[p] <= low[i]) {
                nbcc++;
                is_cut[p] = 1;
                for (int x = 0; x != i; st.pop()) {
                    x = st.top();
                    bcc[nbcc].push_back(x);
                }
                bcc[nbcc].push_back(p);
            } else low[p] = min(low[p], dfn[i]);
        }
    }
}
void build() { // [n+1, n+nbcc] cycle, [1, n] vertex
    for (int i = 1; i <= nbcc; i++) {
        for (auto j : bcc[i]) {
            G[i + n].push_back(j);
            G[j].push_back(i + n);
        }
    }
}

```

## 2.3 BCC Edge

```

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

```

```

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

```

## 2.4 MinimumMeanCycle

```

/* O(V^3)
let dp[i][j] = min length from 1 to j exactly i edges
ans = min (dp[n + 1][u] - dp[i][u]) / (n + 1 - i) */

```

## 2.5 MaximumCliqueDyn

```

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

```

## 2.6 DMST(slow)

```

struct zhu_liu { // O(VE)
    struct edge {
        int u, v;
        ll w;
    };
    vector<edge> E; // 0-base
    int pe[N], id[N], vis[N];
    ll in[N];
    void init() { E.clear(); }
    void add_edge(int u, int v, ll w) {
        if (u != v) E.emplace_back(edge{u, v, w});
    }
    ll build(int root, int n) {
        ll ans = 0;
        for (;;) {
            fill_n(in, n, INF);
            for (int i = 0; i < E.size(); ++i)
                if (E[i].u != E[i].v && E[i].w < in[E[i].v])
                    pe[E[i].v] = i, in[E[i].v] = E[i].w;
            for (int u = 0; u < n; ++u) // no solution
                if (u != root && in[u] == INF) return -INF;
            int cntnode = 0;
            fill_n(id, n, -1), fill_n(vis, n, -1);
            for (int u = 0; u < n; ++u) {
                if (u != root) ans += in[u];
                int v = u;
                while (vis[v] != u && !~id[v] && v != root)
                    vis[v] = u, v = E[pe[v]].u;
                if (v != root && !~id[v]) {
                    for (int x = E[pe[v]].u; x != v;
                        x = E[pe[x]].u)
                        id[x] = cntnode;
                    id[v] = cntnode++;
                }
            }
            if (!cntnode) break; // no cycle
            for (int u = 0; u < n; ++u)
                if (!~id[u]) id[u] = cntnode++;
            for (int i = 0; i < E.size(); ++i) {
                int v = E[i].v;
                E[i].u = id[E[i].u], E[i].v = id[E[i].v];
                if (E[i].u != E[i].v) E[i].w -= in[v];
            }
            n = cntnode, root = id[root];
        }
        return ans;
    }
};

```

## 2.7 DMST

```

#define rep(i, a, b) for (int i = a; i < (b); ++i)
#define all(x) begin(x), end(x)
#define sz(x) (int)(x).size()
typedef vector<int> vi;
struct RollbackUF {
    vi e;
    vector<pii> st;
    RollbackUF(int n) : e(n, -1) {}
    int size(int x) { return -e[find(x)]; }
    int find(int x) { return e[x] < 0 ? x : find(e[x]); }
    int time() { return sz(st); }
    void rollback(int t) {
        for (int i = time(); i-- > t;)
            e[st[i].first] = st[i].second;
        st.resize(t);
    }
    bool join(int a, int b) {
        a = find(a), b = find(b);
        if (a == b) return false;
        if (e[a] > e[b]) swap(a, b);
        st.push_back({a, e[a]});
        st.push_back({b, e[b]});
        e[a] += e[b];
        e[b] = a;
    }
};

```

```

        return true;
    }
};
struct Edge {
    int a, b;
    ll w;
};
struct Node { /// lazy skew heap node
    Edge key;
    Node *l, *r;
    ll delta;
    void prop() {
        key.w += delta;
        if (l) l->delta += delta;
        if (r) r->delta += delta;
        delta = 0;
    }
    Edge top() {
        prop();
        return key;
    }
};
Node *merge(Node *a, Node *b) {
    if (!a || !b) return a ? b : a->prop(), b->prop();
    if (a->key.w > b->key.w) swap(a, b);
    swap(a->l, (a->r = merge(b, a->r)));
    return a;
}
void pop(Node *a) {
    a->prop();
    a = merge(a->l, a->r);
}
pair<ll, vi> dmst(int n, int r, vector<Edge> &g) {
    RollbackUF uf(n);
    vector<Node*> heap(n);
    for (Edge e : g)
        heap[e.b] = merge(heap[e.b], new Node{e});
    ll res = 0;
    vi seen(n, -1), path(n), par(n);
    seen[r] = r;
    vector<Edge> Q(n), in(n, {-1, -1}), comp;
    deque<tuple<int, int, vector<Edge>>> cys;
    rep(s, 0, n) {
        int u = s, qi = 0, w;
        while (seen[u] < 0) {
            if (!heap[u]) return {-1, {}};
            Edge e = heap[u]->top();
            heap[u]->delta -= e.w, pop(heap[u]);
            Q[qi] = e, path[qi++] = u, seen[u] = s;
            res += e.w, u = uf.find(e.a);
            if (seen[u] == s) { /// found cycle, contract
                Node *cyc = 0;
                int end = qi, time = uf.time();
                do cyc = merge(cyc, heap[w = path[--qi]]);
                while (uf.join(u, w));
                u = uf.find(u), heap[u] = cyc, seen[u] = -1;
                cys.push_front({u, time, {&Q[qi], &Q[end]}});
            }
        }
        rep(i, 0, qi) in[uf.find(Q[i].b)] = Q[i];
    }
    for (auto &[u, t, comp] : cys) { /// restore sol (optional)
        uf.rollback(t);
        Edge inEdge = in[u];
        for (auto &e : comp) in[uf.find(e.b)] = e;
        in[uf.find(inEdge.b)] = inEdge;
    }
    rep(i, 0, n) par[i] = in[i].a;
    return {res, par};
}

```

## 2.8 VizingTheorem

```

namespace Vizing { // Edge coloring
                  // G: coloring adjM
int C[N][N], G[N][N];
void clear(int n) {
    for (int i = 0; i <= n; i++) {
        for (int j = 0; j <= n; j++) C[i][j] = G[i][j] = 0;
    }
}
void solve(vector<pii> &E, int n, int m) {
    int X[n] = {}, a;
    auto update = [&](int u) {
        for (X[u] = 1; C[u][X[u]]; X[u]++);
    };
    auto color = [&](int u, int v, int c) {
        int p = G[u][v];
        G[u][v] = G[v][u] = c;
        C[u][c] = v;
        C[v][c] = u;
        C[u][p] = C[v][p] = 0;
        if (p) X[u] = X[v] = p;
        else update(u), update(v);
        return p;
    };
    auto flip = [&](int u, int c1, int c2) {
        int p = C[u][c1];
        swap(C[u][c1], C[u][c2]);
        if (p) G[u][p] = G[p][u] = c2;
        if (!C[u][c1]) X[u] = c1;
        if (!C[u][c2]) X[u] = c2;
        return p;
    };
    for (int i = 1; i <= n; i++) X[i] = 1;
    for (int t = 0; t < E.size(); t++) {
        int u = E[t].first, v0 = E[t].second, v = v0,
            c0 = X[u], c = c0, d;
        vector<pii> L;
        int vst[n] = {};
        while (!G[u][v0]) {
            L.emplace_back(v, d = X[v]);
            if (!C[v][c])
                for (a = (int)L.size() - 1; a >= 0; a--)
                    c = color(u, L[a].first, c);
            else if (!C[u][d])
                for (a = (int)L.size() - 1; a >= 0; a--)
                    color(u, L[a].first, L[a].second);
            else if (vst[d]) break;
            else vst[d] = 1, v = C[u][d];
        }
        if (!G[u][v0]) {
            for (; v; v = flip(v, c, d), swap(c, d));
            if (C[u][c0]) {
                for (a = (int)L.size() - 2;
                    a >= 0 && L[a].second != c; a--)
                    ;
                for (; a >= 0; a--)
                    color(u, L[a].first, L[a].second);
            } else t--;
        }
    }
}
} // namespace Vizing

```

## 2.9 MinimumCliqueCover

```

struct Clique_Cover { // 0-base, O(n^2n)
    int co[1 << N], n, E[N];
    int dp[1 << N];
    void init(int _n) {
        n = _n, fill_n(dp, 1 << n, 0);
        fill_n(E, n, 0), fill_n(co, 1 << n, 0);
    }
    void add_edge(int u, int v) {
        E[u] |= 1 << v, E[v] |= 1 << u;
    }
    int solve() {
        for (int i = 0; i < n; ++i)

```

```

        co[1 << i] = E[i] | (1 << i);
        co[0] = (1 << n) - 1;
        dp[0] = (n & 1) * 2 - 1;
        for (int i = 1; i < (1 << n); ++i) {
            int t = i & -i;
            dp[i] = -dp[i ^ t];
            co[i] = co[i ^ t] & co[t];
        }
        for (int i = 0; i < (1 << n); ++i)
            co[i] = (co[i] & i) == i;
        fwt(co, 1 << n, 1);
        for (int ans = 1; ans < n; ++ans) {
            int sum = 0; // probabilistic
            for (int i = 0; i < (1 << n); ++i)
                sum += (dp[i] * co[i]);
            if (sum) return ans;
        }
        return n;
    }
};

```

## 2.10 CountMaximalClique

```

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

```

## 2.11 Theorems

$|\text{max independent edge set}| = |V| - |\text{min edge cover}|$   
 $|\text{max independent set}| = |V| - |\text{min vertex cover}|$

## 3 Flow-Matching

### 3.1 HopcroftKarp

```

struct hopcroftKarp { // 0-based
    bool dfs(int a, int L, vector<vector<int>> &g,
        vector<int> &btoa, vector<int> &A,
        vector<int> &B) {
        if (A[a] != L) return 0;
        A[a] = -1;
        for (int b : g[a])
            if (B[b] == L + 1) {
                B[b] = 0;

```

```

        if (btoa[b] == -1 ||
            dfs(btoa[b], L + 1, g, btoa, A, B))
            return btoa[b] = a, 1;
    }
    return 0;
}

int solve(vector<vector<int>> &g, int m) {
    int res = 0;
    vector<int> btoa(-1, m), A(g.size()),
    B(btoa.size()), cur, next;
    for (;;) {
        fill(all(A), 0), fill(all(B), 0);
        cur.clear();
        for (int a : btoa)
            if (a != -1) A[a] = -1;
        for (int a = 0; a < g.size(); a++)
            if (A[a] == 0) cur.push_back(a);
        /// Find all layers using bfs.
        for (int lay = 1; lay++ < g.size()) {
            bool islast = 0;
            next.clear();
            for (int a : cur)
                for (int b : g[a]) {
                    if (btoa[b] == -1) {
                        B[b] = lay;
                        islast = 1;
                    } else if (btoa[b] != a && !B[b]) {
                        B[b] = lay;
                        next.push_back(btoa[b]);
                    }
                }
            if (islast) break;
            if (next.empty()) return res;
            for (int a : next) A[a] = lay;
            cur.swap(next);
        }
        /// Use DFS to scan for augmenting paths.
        for (int a = 0; a < g.size(); a++)
            res += dfs(a, 0, g, btoa, A, B);
    }
}
};

```

### 3.2 KM

```

struct KM { /// 0-base
    ll w[N][N], hl[N], hr[N], slk[N];
    int fl[N], fr[N], pre[N], qu[N], ql, qr, n;
    bool vl[N], vr[N];
    void init(int _n) {
        n = _n;
        for (int i = 0; i < n; ++i)
            fill_n(w[i], n, -INF);
    }
    void add_edge(int a, int b, ll wei) {
        w[a][b] = wei;
    }
    bool Check(int x) {
        if (vl[x] = 1, ~fl[x])
            return vr[qu[qr++]] = fl[x] = 1;
        while (~x) swap(x, fr[fl[x] = pre[x]]);
        return 0;
    }
    void bfs(int s) {
        fill_n(slk,
            , n, INF), fill_n(vl, n, 0), fill_n(vr, n, 0);
        ql = qr = 0, qu[qr++] = s, vr[s] = 1;
        for (ll d; ; ) {
            while (ql < qr)
                for (int x = 0, y = qu[ql++]; x < n; ++x)
                    if (!vl[x] && slk
                        [x] >= (d = hl[x] + hr[y] - w[x][y])) {
                        if (pre[x] = y, d) slk[x] = d;
                        else if (!Check(x)) return;
                    }
            }
        }
    }
};

```

```

d = INF;
for (int x = 0; x < n; ++x)
    if (!vl[x] && d > slk[x]) d = slk[x];
for (int x = 0; x < n; ++x) {
    if (vl[x]) hl[x] += d;
    else slk[x] -= d;
    if (vr[x]) hr[x] -= d;
}
for (int x = 0; x < n; ++x)
    if (!vl[x] && !slk[x] && !Check(x)) return;
}
}

ll solve() {
    fill_n(fl,
        , n, -1), fill_n(fr, n, -1), fill_n(hr, n, 0);
    for (int i = 0; i < n; ++i)
        hl[i] = *max_element(w[i], w[i] + n);
    for (int i = 0; i < n; ++i) bfs(i);
    ll res = 0;
    for (int i = 0; i < n; ++i) res += w[i][fl[i]];
    return res;
}
};

```

### 3.3 MCMF

```

struct MinCostMaxFlow { /// 0-base
    struct Edge {
        ll from, to, cap, flow, cost, rev;
    } *past[N];
    vector<Edge> G[N];
    int inq[N], n, s, t;
    ll dis[N], up[N], pot[N];
    bool BellmanFord() {
        fill_n(dis, n, INF), fill_n(inq, n, 0);
        queue<int> q;
        auto relax = [&](int u, ll d, ll cap, Edge *e) {
            if (cap > 0 && dis[u] > d) {
                dis[u] = d, up[u] = cap, past[u] = e;
                if (!inq[u]) inq[u] = 1, q.push(u);
            }
        };
        relax(s, 0, INF, 0);
        while (!q.empty()) {
            int u = q.front();
            q.pop(), inq[u] = 0;
            for (auto &e : G[u]) {
                ll d2 = dis[u] + e.cost + pot[u] - pot[e.to];
                relax
                    (e.to, d2, min(up[u], e.cap - e.flow), &e);
            }
        }
        return dis[t] != INF;
    }

    void solve(int _s
        , int _t, ll &flow, ll &cost, bool neg = true) {
        s = _s, t = _t, flow = 0, cost = 0;
        if (neg) BellmanFord(), copy_n(dis, n, pot);
        for (; BellmanFord(); copy_n(dis, n, pot)) {
            for (int
                i = 0; i < n; ++i) dis[i] += pot[i] - pot[s];
            flow += up[t], cost += up[t] * dis[t];
            for (int i = t; past[i]; i = past[i]->from) {
                auto &e = *past[i];
                e.flow += up[t], G[e.to][e.rev].flow -= up[t];
            }
        }
    }

    void init(int _n) {
        n = _n, fill_n(pot, n, 0);
        for (int i = 0; i < n; ++i) G[i].clear();
    }

    void add_edge(ll a, ll b, ll cap, ll cost) {
        G[a].emplace_back
            (Edge{a, b, cap, 0, cost, (ll)G[b].size()});
        G[b].emplace_back
            (Edge{b, a, 0, 0, -cost, (ll)G[a].size() - 1});
    }
};

```

```

}
};

```

### 3.4 GeneralGraphMatching

```

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

```

### 3.5 MaxWeightMaching

```

#define rep(i, l, r) for (int i = (l); i <= (r); ++i)
struct WeightGraph { // 1-based
    struct edge {
        int u, v, w;
    };
    int n, nx;
    vector<int> lab;
    vector<vector<edge>> g;
    vector<int> slack, match, st, pa, S, vis;
    vector<vector<int>> flo, flo_from;
    queue<int> q;
    WeightGraph(int n_)
        : n(n_), nx(n * 2), lab(nx + 1),
          g(nx + 1, vector<edge>(nx + 1)), slack(nx + 1),
          flo(nx + 1), flo_from(nx + 1, vector<int>(n + 1, 0)) {
        match = st = pa = S = vis = slack;
        rep(u, 1, n) rep(v, 1, n) g[u][v] = {u, v, 0};
    }

```

```

}
int ED(edge e) {
    return lab[e.u] + lab[e.v] - g[e.u][e.v].w * 2;
}
void update_slack(int u, int x, int &s) {
    if (!s || ED(g[u][x]) < ED(g[s][x])) s = u;
}
void set_slack(int x) {
    slack[x] = 0;
    for (int u = 1; u <= n; ++u)
        if (g[u][x].w > 0 && st[u] != x && S[st[u]] == 0)
            update_slack(u, x, slack[x]);
}
void q_push(int x) {
    if (x <= n) q.push(x);
    else
        for (int y: flo[x]) q.push(y);
}
void set_st(int x, int b) {
    st[x] = b;
    if (x > n)
        for (int y: flo[x]) set_st(y, b);
}
vector<int> split_flo(auto &f, int xr) {
    auto it = find(ALL(f), xr);
    if (auto pr = it - f.begin(); pr % 2 == 1)
        reverse(1 + ALL(f), it = f.end() - pr);
    auto res = vector<int>(f.begin(), it);
    return f.erase(f.begin(), it), res;
}
void set_match(int u, int v) {
    match[u] = g[u][v].v;
    if (u <= n) return;
    int xr = flo_from[u][g[u][v].u];
    auto &f = flo[u], z = split_flo(f, xr);
    rep(i, 0, (int)z.size() - 1)
        set_match(z[i], z[i ^ 1]);
    set_match(xr, v);
    f.insert(f.end(), all(z));
}
void augment(int u, int v) {
    for (;;) {
        int xnv = st[match[u]];
        set_match(u, v);
        if (!xnv) return;
        set_match(xnv, st[pa[xnv]]);
        u = st[pa[xnv]], v = xnv;
    }
}
int lca(int u, int v) {
    static int t = 0;
    ++t;
    for (++t; u || v; swap(u, v))
        if (u) {
            if (vis[u] == t) return u;
            vis[u] = t;
            u = st[match[u]];
            if (u) u = st[pa[u]];
        }
    return 0;
}
void add_blossom(int u, int o, int v) {
    int b = find(n + 1 + all(st), 0) - begin(st);
    lab[b] = 0, S[b] = 0;
    match[b] = match[o];
    vector<int> f = {o};
    for (int x = u, y; x != o; x = st[pa[y]])
        f.emplace_back(x),
        f.emplace_back(y = st[match[x]]), q.push(y);
    reverse(1 + all(f));
    for (int x = v, y; x != o; x = st[pa[y]])
        f.emplace_back(x),
        f.emplace_back(y = st[match[x]]), q.push(y);
    flo[b] = f;
    set_st(b, b);
}

```



```

for (int x = 1; x <= nx; ++x)
    g[b][x].w = g[x][b].w = 0;
fill(all(flo_from[b]), 0);
for (int xs : flo[b]) {
    for (int x = 1; x <= nx; ++x)
        if (g[b][x].w == 0 ||
            ED(g[xs][x]) < ED(g[b][x]))
            g[b][x] = g[xs][x], g[x][b] = g[x][xs];
    for (int x = 1; x <= n; ++x)
        if (flo_from[xs][x]) flo_from[b][x] = xs;
}
set_slack(b);
}
void expand_blossom(int b) {
    for (int x : flo[b]) set_st(x, x);
    int xr = flo_from[b][g[b][pa[b]].u], xs = -1;
    for (int x : split_flo(flo[b], xr)) {
        if (xs == -1) {
            xs = x;
            continue;
        }
        pa[xs] = g[x][xs].u;
        S[xs] = 1, S[x] = 0;
        slack[xs] = 0;
        set_slack(x);
        q_push(x);
        xs = -1;
    }
    for (int x : flo[b])
        if (x == xr) S[x] = 1, pa[x] = pa[b];
        else S[x] = -1, set_slack(x);
    st[b] = 0;
}
bool on_found_edge(const edge &e) {
    if (int u = st[e.u], v = st[e.v]; S[v] == -1) {
        int nu = st[match[v]];
        pa[v] = e.u;
        S[v] = 1;
        slack[v] = slack[nu] = 0;
        S[nu] = 0;
        q_push(nu);
    } else if (S[v] == 0) {
        if (int o = lca(u, v)) add_blossom(u, o, v);
        else return augment(u, v), augment(v, u), true;
    }
    return false;
}
}
bool matching() {
    fill(all(S), -1), fill(all(slack), 0);
    q = queue<int>();
    for (int x = 1; x <= nx; ++x)
        if (st[x] == x && !match[x])
            pa[x] = 0, S[x] = 0, q_push(x);
    if (q.empty()) return false;
    for (;;) {
        while (q.size()) {
            int u = q.front();
            q.pop();
            if (S[st[u]] == 1) continue;
            for (int v = 1; v <= n; ++v)
                if (g[u][v].w > 0 && st[u] != st[v]) {
                    if (ED(g[u][v]) != 0)
                        update_slack(u, st[v], slack[st[v]]);
                    else if (on_found_edge(g[u][v]))
                        return true;
                }
        }
        int d = INF;
        for (int b = n + 1; b <= nx; ++b)
            if (st[b] == b && S[b] == 1)
                d = min(d, lab[b] / 2);
        for (int x = 1; x <= nx; ++x)
            if (int s = slack[x];
                st[x] == x && s && S[x] <= 0)
                d = min(d, ED(g[s][x]) / (S[x] + 2));
    }
}

```

```

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

```

### 3.6 GlobalMinCut

```

#undef INF
struct SW{ // global min cut,  $O(V^3)$ 
#define REP for (int i = 0; i < n; ++i)
static const int MXN = 514, INF = 2147483647;
int vst[MXN], edge[MXN][MXN], wei[MXN];
void init(int n) {
    REP fill_n(edge[i], n, 0);
}
void addEdge(int u, int v, int w){
    edge[u][v] += w; edge[v][u] += w;
}
int search(int &s, int &t, int n){
    fill_n(vst, n, 0), fill_n(wei, n, 0);
    s = t = -1;
    int mx, cur;
    for (int j = 0; j < n; ++j) {
        mx = -1, cur = 0;
        REP if (wei[i] > mx) cur = i, mx = wei[i];
        vst[cur] = 1, wei[cur] = -1;
        s = t; t = cur;
        REP if (!vst[i]) wei[i] += edge[cur][i];
    }
    return mx;
}
int solve(int n) {
    int res = INF;
    for (int x, y; n > 1; n--){
        res = min(res, search(x, y, n));
        REP edge[i][x] = (edge[x][i] += edge[y][i]);
        REP {
            edge[y][i] = edge[n - 1][i];
            edge[i][y] = edge[i][n - 1];
        } // edge[y][y] = 0;
    }
    return res;
}
}

```

```
} sw;
```

### 3.7 BoundedFlow(Dinic)

```
struct BoundedFlow { // 0-base
    struct edge {
        int to, cap, flow, rev;
    };
    vector<edge> G[N];
    int n, s, t, dis[N], cur[N], cnt[N];
    void init(int _n) {
        n = _n;
        for (int i = 0; i < n + 2; ++i)
            G[i].clear(), cnt[i] = 0;
    }
    void add_edge(int u, int v, int lcap, int rcap) {
        cnt[u] -= lcap, cnt[v] += lcap;
        G[u].emplace_back
            (edge{v, rcap, lcap, (int)G[v].size()});
        G[v].emplace_back
            (edge{u, 0, 0, (int)G[u].size() - 1});
    }
    void add_edge(int u, int v, int cap) {
        G[u].emplace_back
            (edge{v, cap, 0, (int)G[v].size()});
        G[v].emplace_back
            (edge{u, 0, 0, (int)G[u].size() - 1});
    }
    int dfs(int u, int cap) {
        if (u == t || !cap) return cap;
        for (int &i = cur[u]; i < G[u].size(); ++i) {
            edge &e = G[u][i];
            if (dis[e.to] == dis[u] + 1 && e.cap != e.flow) {
                int df = dfs(e.to, min(e.cap - e.flow, cap));
                if (df) {
                    e.flow += df, G[e.to][e.rev].flow -= df;
                    return df;
                }
            }
        }
        dis[u] = -1;
        return 0;
    }
    bool bfs() {
        fill_n(dis, n + 3, -1);
        queue<int> q;
        q.push(s), dis[s] = 0;
        while (!q.empty()) {
            int u = q.front();
            q.pop();
            for (edge &e : G[u])
                if (!dis[e.to] && e.flow != e.cap)
                    q.push(e.to), dis[e.to] = dis[u] + 1;
        }
        return dis[t] != -1;
    }
    int maxflow(int _s, int _t) {
        s = _s, t = _t;
        int flow = 0, df;
        while (bfs()) {
            fill_n(cur, n + 3, 0);
            while ((df = dfs(s, INF))) flow += df;
        }
        return flow;
    }
    bool solve() {
        int sum = 0;
        for (int i = 0; i < n; ++i)
            if (cnt[i] > 0)
                add_edge(n + 1, i, cnt[i]), sum += cnt[i];
            else if (cnt[i] < 0) add_edge(i, n + 2, -cnt[i]);
        if (sum != maxflow(n + 1, n + 2)) sum = -1;
        for (int i = 0; i < n; ++i)
            if (cnt[i] > 0)
                G[n + 1].pop_back(), G[i].pop_back();
            else if (cnt[i] < 0)
                G[i].pop_back(), G[n + 2].pop_back();
    }
};
```

```
        return sum != -1;
    }
    int solve(int _s, int _t) {
        add_edge(_t, _s, INF);
        if (!solve()) return -1; // invalid flow
        int x = G[_t].back().flow;
        return G[_t].pop_back(), G[_s].pop_back(), x;
    }
};
```

### 3.8 GomoryHuTree

```
MaxFlow Dinic;
int g[N];
void GomoryHu(int n) { // 0-base
    fill_n(g, n, 0);
    for (int i = 1; i < n; ++i) {
        Dinic.reset();
        add_edge(i, g[i], Dinic.maxflow(i, g[i]));
        for (int j = i + 1; j <= n; ++j)
            if (g[j] == g[i] && ~Dinic.dis[j])
                g[j] = i;
    }
}
```

### 3.9 MinCostCirculation

```
struct MinCostCirculation { // 0-base
    struct Edge {
        ll from, to, cap, fcap, flow, cost, rev;
    } *past[N];
    vector<Edge> G[N];
    ll dis[N], inq[N], n;
    void BellmanFord(int s) {
        fill_n(dis, n, INF), fill_n(inq, n, 0);
        queue<int> q;
        auto relax = [&](int u, ll d, Edge *e) {
            if (dis[u] > d) {
                dis[u] = d, past[u] = e;
                if (!inq[u]) inq[u] = 1, q.push(u);
            }
        };
        relax(s, 0, 0);
        while (!q.empty()) {
            int u = q.front();
            q.pop(), inq[u] = 0;
            for (auto &e : G[u])
                if (e.cap > e.flow)
                    relax(e.to, dis[u] + e.cost, &e);
        }
    }
    void try_edge(Edge &cur) {
        if (cur.cap > cur.flow) return ++cur.cap, void();
        BellmanFord(cur.to);
        if (dis[cur.from] + cur.cost < 0) {
            ++cur.flow, --G[cur.to][cur.rev].flow;
            for (int
                i = cur.from; past[i]; i = past[i]->from) {
                auto &e = *past[i];
                ++e.flow, --G[e.to][e.rev].flow;
            }
        }
        ++cur.cap;
    }
    void solve(int mxlg) {
        for (int b = mxlg; b >= 0; --b) {
            for (int i = 0; i < n; ++i)
                for (auto &e : G[i])
                    e.cap *= 2, e.flow *= 2;
            for (int i = 0; i < n; ++i)
                for (auto &e : G[i])
                    if (e.fcap >> b & 1)
                        try_edge(e);
        }
    }
    void init(int _n) { n = _n;
        for (int i = 0; i < n; ++i) G[i].clear();
    }
};
```



```

}
void add_edge(ll a, ll b, ll cap, ll cost) {
    G[a].emplace_back(Edge{a, b,
        0, cap, 0, cost, (ll)G[b].size() + (a == b)});
    G[b].emplace_back(Edge
        {b, a, 0, 0, 0, -cost, (ll)G[a].size() - 1});
}
} mcmf; // O(VE * ElogC)

```

### 3.10 FlowModelsBuilding

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

$$\begin{aligned}
 \min \sum_{uv} w_{uv} f_{uv} & \quad \min \sum_u b_u p_u + \sum_{uv} c_{uv} \max(0, p_v - p_u - w_{uv}) \\
 -f_{uv} & \geq -c_{uv} \Leftrightarrow & p_u \geq 0 \\
 \sum_v f_{vu} - \sum_v f_{uv} & = -b_u
 \end{aligned}$$

## 4 Data Structure

### 4.1 LCT

```

#define ls(x) Tree[x].son[0]
#define rs(x) Tree[x].son[1]
#define fa(x) Tree[x].fa
const int maxn = 600010;
struct node {
    int son[2], Min, id, fa, lazy;
} Tree[maxn];
int n, m, q, w[maxn], Min;
struct Node {
    int u, v, w;
} a[maxn];
inline bool IsRoot(int x) {
    return (ls(fa(x)) == x || rs(fa(x)) == x) ? false : true;
}
inline void PushUp(int x) {
    Tree[x].Min = w[x], Tree[x].id = x;
    if (ls(x) && Tree[ls(x)].Min < Tree[x].Min) {
        Tree[x].Min = Tree[ls(x)].Min;
        Tree[x].id = Tree[ls(x)].id;
    }
    if (rs(x) && Tree[rs(x)].Min < Tree[x].Min) {
        Tree[x].Min = Tree[rs(x)].Min;
        Tree[x].id = Tree[rs(x)].id;
    }
}
inline void Update(int x) {
    Tree[x].lazy ^= 1;
    swap(ls(x), rs(x));
}
inline void PushDown(int x) {
    if (!Tree[x].lazy) return;
    if (ls(x)) Update(ls(x));
    if (rs(x)) Update(rs(x));
    Tree[x].lazy = 0;
}
inline void Rotate(int x) {
    int y = fa(x), z = fa(y), k = rs(y) == x,
        w = Tree[x].son[!k];
    if (!IsRoot(y)) Tree[z].son[rs(z) == y] = x;
    fa(x) = z, fa(y) = x;
    if (w) fa(w) = y;
    Tree[x].son[!k] = y, Tree[y].son[k] = w;
    PushUp(y);
}
inline void Splay(int x) {
    stack<int> Stack;
    int y = x, z;
    Stack.push(y);
    while (!IsRoot(y)) Stack.push(y = fa(y));
    while (!Stack.empty())
        PushDown(Stack.top()), Stack.pop();
    while (!IsRoot(x)) {
        y = fa(x), z = fa(y);
        if (!IsRoot(y))
            Rotate((ls(y) == x) ^ (ls(z) == y) ? x : y);
        Rotate(x);
    }
    PushUp(x);
}
inline void Access(int root) {
    for (int x = 0; root; x = root, root = fa(root))
        Splay(root), rs(root) = x, PushUp(root);
}
inline void MakeRoot(int x) {
    Access(x), Splay(x), Update(x);
}
inline int FindRoot(int x) {
    Access(x), Splay(x);
    while (ls(x)) x = ls(x);
    return Splay(x), x;
}

```

```

inline void Link(int u, int v) {
    MakeRoot(u);
    if (FindRoot(v) != u) fa(u) = v;
}
inline void Cut(int u, int v) {
    MakeRoot(u);
    if (FindRoot(v) != u || fa(v) != u || ls(v)) return;
    fa(v) = rs(u) = 0;
}
inline void Split(int u, int v) {
    MakeRoot(u), Access(v), Splay(v);
}
inline bool Check(int u, int v) {
    return MakeRoot(u), FindRoot(v) == u;
}
inline int LCA(int root, int u, int v) {
    MakeRoot(root), Access(u), Access(v), Splay(u);
    if (!fa(u)) {
        Access(u), Splay(v);
        return fa(v);
    }
    return fa(u);
}
/* ETT
每次進入節點和走邊都放入一次共 3n - 2
node(u) 表示進入節點 u 放入 treap 的位置
edge(u, v) 表示 u -> v 的邊放入 treap 的位置 (push v)
Makeroot u :
    L1 = [begin, node(u) - 1], L2 = [node(u), end]
    -> L2 + L1
Insert u, v :
    Tu -> L1 = [begin, node(u) - 1], L2 = [node(u), end]
    Tv -> L3 = [begin, node(v) - 1], L4 = [node(v), end]
    -> L2 + L1 + edge(u, v) + L4 + L3 + edge(v, u)
Delect u, v :
    maybe need swap u, v
    T -> L1 + edge(u, v) + L2 + edge(v, u) + L3
    -> L1 + L3, L2
*/

```

## 4.2 Treap

```

struct node {
    int data, sz;
    node *l, *r;
    node(int k) : data(k), sz(1), l(0), r(0) {}
    void up() {
        sz = 1;
        if (l) sz += l->sz;
        if (r) sz += r->sz;
    }
    void down() {}
};
int sz(node *a) { return a ? a->sz : 0; }
node *merge(node *a, node *b) {
    if (!a || !b) return a ? a : b;
    if (rand() % (sz(a) + sz(b)) < sz(a))
        return a->down(), a->r = merge(a->r, b), a->up(),
            a;
    return b->down(), b->l = merge(a, b->l), b->up(), b;
}
void split(node *o, node *&a, node *&b, int k) {
    if (!o) return a = b = 0, void();
    o->down();
    if (o->data <= k)
        a = o, split(o->r, a->r, b, k), a->up();
    else b = o, split(o->l, a, b->l, k), b->up();
}
void split2(node *o, node *&a, node *&b, int k) {
    if (sz(o) <= k) return a = o, b = 0, void();
    o->down();
    if (sz(o->l) + 1 <= k)
        a = o, split2(o->r, a->r, b, k - sz(o->l) - 1);
    else b = o, split2(o->l, a, b->l, k);
    o->up();
}

```

```

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

```

## 4.3 CentroidDecomposition

```

struct Cent_Dec { // 1-base
    vector<pll> G[N];
    pll info[N]; // store info. of itself
    pll upinfo[N]; // store info. of climbing up
    int n, pa[N], layer[N], sz[N], done[N];
    ll dis[__lg(N) + 1][N];
    void init(int _n) {
        n = _n, layer[0] = -1;
        fill_n(pa + 1, n, 0), fill_n(done + 1, n, 0);
        for (int i = 1; i <= n; ++i) G[i].clear();
    }
    void add_edge(int a, int b, int w) {
        G[a].pb(pll(b, w)), G[b].pb(pll(a, w));
    }
    void get_cent(
        int u, int f, int &mx, int &c, int num) {
        int mxsz = 0;
        sz[u] = 1;
        for (pll e : G[u])
            if (!done[e.X] && e.X != f) {
                get_cent(e.X, u, mx, c, num);
                sz[u] += sz[e.X], mxsz = max(mxsz, sz[e.X]);
            }
        if (mx > max(mxsz, num - sz[u]))
            mx = max(mxsz, num - sz[u]), c = u;
    }
    void dfs(int u, int f, ll d, int org) {
        // if required, add self info or climbing info
        dis[layer[org]][u] = d;
        for (pll e : G[u])
            if (!done[e.X] && e.X != f)
                dfs(e.X, u, d + e.Y, org);
    }
    int cut(int u, int f, int num) {
        int mx = 1e9, c = 0, lc;
        get_cent(u, f, mx, c, num);
        done[c] = 1, pa[c] = f, layer[c] = layer[f] + 1;
        for (pll e : G[c])
            if (!done[e.X]) {
                if (sz[e.X] > sz[c])

```

```

        lc = cut(e.X, c, num - sz[c]);
    else lc = cut(e.X, c, sz[e.X]);
    upinfo[lc] = pll(), dfs(e.X, c, e.Y, c);
}
return done[c] = 0, c;
}
void build() { cut(1, 0, n); }
void modify(int u) {
    for (int a = u, ly = layer[a]; a;
        a = pa[a], --ly) {
        info[a].X += dis[ly][u], ++info[a].Y;
        if (pa[a])
            upinfo[a].X += dis[ly - 1][u], ++upinfo[a].Y;
    }
}
ll query(int u) {
    ll rt = 0;
    for (int a = u, ly = layer[a]; a;
        a = pa[a], --ly) {
        rt += info[a].X + info[a].Y * dis[ly][u];
        if (pa[a])
            rt -=
                upinfo[a].X + upinfo[a].Y * dis[ly - 1][u];
    }
    return rt;
}
};

```

#### 4.4 HeavyLightDecomposition

```

struct Heavy_light_Decomposition { // 1-base
    int n, ulink[N], deep[N], mxson[N], w[N], pa[N];
    int t, pl[N], data[N], dt[N], bln[N], edge[N], et;
    vector<pii> G[N];
    void init(int _n) {
        n = _n, t = 0, et = 1;
        for (int i = 1; i <= n; ++i)
            G[i].clear(), mxson[i] = 0;
    }
    void add_edge(int a, int b, int w) {
        G[a].pb(pii(b, et));
        G[b].pb(pii(a, et));
        edge[et++] = w;
    }
    void dfs(int u, int f, int d) {
        w[u] = 1, pa[u] = f, deep[u] = d++;
        for (auto &i : G[u])
            if (i.X != f) {
                dfs(i.X, u, d), w[u] += w[i.X];
                if (w[mxson[u]] < w[i.X]) mxson[u] = i.X;
                else bln[i.Y] = u, dt[u] = edge[i.Y];
            }
    }
    void cut(int u, int link) {
        data[pl[u] = t++] = dt[u], ulink[u] = link;
        if (!mxson[u]) return;
        cut(mxson[u], link);
        for (auto i : G[u])
            if (i.X != pa[u] && i.X != mxson[u])
                cut(i.X, i.X);
    }
    void build() { dfs(1, 1, 1), cut(1, 1), /*build*/; }
    int query(int a, int b) {
        int ta = ulink[a], tb = ulink[b], re = 0;
        while (ta != tb)
            if (deep[ta] < deep[tb])
                /*query*/, tb = ulink[b = pa[tb]];
            else /*query*/, ta = ulink[a = pa[ta]];
        if (a == b) return re;
        if (pl[a] > pl[b]) swap(a, b);
        /*query*/
        return re;
    }
};

```

## 5 String

### 5.1 KMP

```

int KMP(string s, string t) {
    t = " " + s + t; // consistency with ACa
    int n = t.size(), ans = 0;
    vector<int> f(t.size(), 0);
    f[0] = -1;
    for (int i = 1, j = -1; i < t.size(); i++) {
        while (j >= 0 && t[j + 1] != t[i])
            j = f[j];
        f[i] = ++j;
    }
    for (int i = 0, j = 0; i < s.size(); i++) {
        while (j >= 0 && t[j + 1] != s[i])
            j = f[j];
        if (++j + 1 == t.size()) ans++, j = f[j];
    }
    return ans;
}

```

### 5.2 Z

```

int Z[10000006];
void z(string s) {
    for (int i = 1, mx = 0; i < s.size(); i++) {
        if (i < Z[mx] + mx)
            Z[i] = min(Z[mx] - i + mx, Z[i - mx]);
        while (
            Z[i] + i < s.size() && s[i + Z[i]] == s[Z[i]])
            Z[i]++;
        if (Z[i] + i > Z[mx] + mx) mx = i;
    }
}

```

### 5.3 Manacher

```

int man[20000006];
int manacher(string s) {
    string t;
    for (int i = 0; i < s.size(); i++) {
        if (i) t.push_back('$');
        t.push_back(s[i]);
    }
    int mx = 0, ans = 0;
    for (int i = 0; i < t.size(); i++) {
        man[i] = 1;
        man[i] = min(man[mx] + mx - i, man[2 * mx - i]);
        while (man[i] + i < t.size() && i - man[i] >= 0 &&
            t[i + man[i]] == t[i - man[i]])
            man[i]++;
        if (i + man[i] > mx + man[mx]) mx = i;
    }
    for (int i = 0; i < t.size(); i++)
        ans = max(ans, man[i] - 1);
    return ans;
}

```

### 5.4 SuffixArray

```

vector<int> sa, cnt, rk, tmp, lcp;
void SA(string s) {
    int n = s.size();
    sa.resize(n), cnt.resize(n), rk.resize(n),
        tmp.resize(n);
    iota(all(sa), 0);
    sort(all(sa),
        [&](int i, int j) { return s[i] < s[j]; });
    rk[0] = 0;
    for (int i = 1; i < n; i++)
        rk[sa[i]] =
            rk[sa[i - 1]] + (s[sa[i - 1]] != s[sa[i]]);
    for (int k = 1; k <= n; k <= 1) {
        fill(all(cnt), 0);
        for (int i = 0; i < n; i++)
            cnt[rk[(sa[i] - k + n) % n]]++;
        for (int i = 1; i < n; i++) cnt[i] += cnt[i - 1];
        for (int i = n - 1; i >= 0; i--)
            tmp[--cnt[rk[(sa[i] - k + n) % n]]] =
                (sa[i] - k + n) % n;
    }
}

```

```

    sa.swap(tmp);
    tmp[sa[0]] = 0;
    for (int i = 1; i < n; i++)
        tmp[sa[i]] = tmp[sa[i - 1]] +
            (rk[sa[i - 1]] != rk[sa[i]] ||
             rk[(sa[i - 1] + k) % n] !=
             rk[(sa[i] + k) % n]);
    rk.swap(tmp);
}

void LCP(string s) {
    int n = s.size(), k = 0;
    lcp.resize(n);
    for (int i = 0; i < n; i++)
        if (rk[i] == 0) lcp[rk[i]] = 0;
        else {
            if (k) k--;
            int j = sa[rk[i] - 1];
            while (
                i + k < n && j + k < n && s[i + k] == s[j + k])
                k++;
            lcp[rk[i]] = k;
        }
}

```

## 5.5 SAIS

```

namespace sfx {
bool _t[N * 2];
int SA[N * 2], H[N], RA[N];
int _s[N * 2], _c[N * 2], x[N], _p[N], _q[N * 2];
// zero based, string content MUST > 0
// SA[i]: SA[i]-th
//          suffix is the i-th lexicographically smallest suffix.
// H[i]: longest
//          common prefix of suffix SA[i] and suffix SA[i - 1].
void pre(int *sa, int *c, int n, int z)
{ fill_n(sa, n, 0), copy_n(c, z, x); }
void induce
    (int *sa, int *c, int *s, bool *t, int n, int z) {
    copy_n(c, z - 1, x + 1);
    for (int i = 0; i < n; ++i)
        if (sa[i] && !t[sa[i] - 1])
            sa[x[s[sa[i] - 1]]++] = sa[i] - 1;
    copy_n(c, z, x);
    for (int i = n - 1; i >= 0; --i)
        if (sa[i] && t[sa[i] - 1])
            sa[--x[s[sa[i] - 1]]] = sa[i] - 1;
}

void sais(int *s, int *sa
    , int *p, int *q, bool *t, int *c, int n, int z) {
    bool uniq = t[n - 1] = true;
    int nn = 0,
        nmzx = -1, *nsa = sa + n, *ns = s + n, last = -1;
    fill_n(c, z, 0);
    for (int i = 0; i < n; ++i) uniq &= ++c[s[i]] < 2;
    partial_sum(c, c + z, c);
    if (uniq) {
        for (int i = 0; i < n; ++i) sa[--c[s[i]]] = i;
        return;
    }
    for (int i = n - 2; i >= 0; --i)
        t[i] = (
            s[i] == s[i + 1] ? t[i + 1] : s[i] < s[i + 1]);
    pre(sa, c, n, z);
    for (int i = 1; i <= n - 1; ++i)
        if (t[i] && !t[i - 1])
            sa[--x[s[i]]] = p[q[i] = nn++] = i;
    induce(sa, c, s, t, n, z);
    for (int i = 0; i < n; ++i)
        if (sa[i] && t[sa[i]] && !t[sa[i] - 1]) {
            bool neq = last < 0 || !equal
                (s + sa[i], s + p[q[sa[i]] + 1], s + last);
            ns[q[last = sa[i]]] = nmzx += neq;
        }
}

```

```

sais(ns,
    nsa, p + nn, q + n, t + n, c + z, nn, nmzx + 1);
pre(sa, c, n, z);
for (int i = nn - 1; i >= 0; --i)
    sa[--x[s[p[nsa[i]]]]] = p[nsa[i]];
induce(sa, c, s, t, n, z);
}

void mkhei(int n) {
    for (int i = 0, j = 0; i < n; ++i) {
        if (RA[i])
            for (; _s[i + j] == _s[SA[RA[i] - 1] + j]; ++j);
        H[RA[i]] = j, j = max(0, j - 1);
    }
}

void build(int *s, int n) {
    copy_n(s, n, _s), _s[n] = 0;
    sais(_s, SA, _p, _q, _t, _c, n + 1, 256);
    copy_n(SA + 1, n, SA);
    for (int i = 0; i < n; ++i) RA[SA[i]] = i;
    mkhei(n);
}

```

## 5.6 ACAutomaton

```

#define sumS 500005
#define sigma 26
#define base 'a'
struct AhoCorasick {
    int ch[sumS][sigma] = {{}}, f[sumS] = {-1},
        tag[sumS], mv[sumS][sigma], jump[sumS],
        cnt[sumS];
    int idx = 0, t = -1;
    vector<int> E[sumS], q;
    pii o[sumS];
    int insert(string &s, int t) {
        int j = 0;
        for (int i = 0; i < (int)s.size(); i++) {
            if (!ch[j][s[i] - base])
                ch[j][s[i] - base] = ++idx;
            j = ch[j][s[i] - base];
        }
        tag[j] = 1;
        return j;
    }

    int next(int u, int c) {
        return u < 0 ? 0 : mv[u][c];
    }

    void dfs(int u) {
        o[u].F = ++t;
        for (auto v : E[u]) dfs(v);
        o[u].S = t;
    }

    void build() {
        int k = -1;
        q.emplace_back(0);
        while (++k < q.size()) {
            int u = q[k];
            for (int v = 0; v < sigma; v++) {
                if (ch[u][v]) {
                    f[ch[u][v]] = next(f[u], v);
                    q.emplace_back(ch[u][v]);
                }
                mv[u][v] =
                    (ch[u][v] ? ch[u][v] : next(f[u], v));
            }
            if (u) jump[u] = (tag[f[u]] ? f[u] : jump[f[u]]);
        }
        reverse(q.begin(), q.end());
        for (int i = 1; i <= idx; i++)
            E[f[i]].emplace_back(i);
        dfs(0);
    }

    void match(string &s) {
        fill(cnt, cnt + idx + 1, 0);
        for (int i = 0, j = 0; i < (int)s.size(); i++)
            cnt[j = next(j, s[i] - base)]++;
    }
}

```

```

    for (int i : q)
        if (f[i] > 0) cnt[f[i]] += cnt[i];
}
} ac;

```

## 5.7 MinRotation

```

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

```

## 5.8 ExtSAM

```

#define CNUM 26
struct exSAM {
    int len[N * 2], link[N * 2]; // maxlength, suf link
    int next[N * 2][CNUM], tot; // [0, tot), root = 0
    int lenSorted[N * 2]; // topo. order
    int cnt[N * 2]; // occurrence
    int newnode() {
        fill_n(next[tot], CNUM, 0);
        len[tot] = cnt[tot] = link[tot] = 0;
        return tot++;
    }
    void init() { tot = 0, newnode(), link[0] = -1; }
    int insertSAM(int last, int c) {
        int cur = next[last][c];
        len[cur] = len[last] + 1;
        int p = link[last];
        while (p != -1 && !next[p][c])
            next[p][c] = cur, p = link[p];
        if (p == -1) return link[cur] = 0, cur;
        int q = next[p][c];
        if (len[p] + 1 == len[q]) return link[cur] = q, cur;
        int clone = newnode();
        for (int i = 0; i < CNUM; ++i)
            next[clone][i] = len[next[q][i]] ? next[q][i] : 0;
        len[clone] = len[p] + 1;
        while (p != -1 && next[p][c] == q)
            next[p][c] = clone, p = link[p];
        link[link[cur]] = clone;
        link[q] = clone;
        return cur;
    }
    void insert(const string &s) {
        int cur = 0;
        for (auto ch : s) {
            int &nxt = next[cur][int(ch - 'a')];
            if (!nxt) nxt = newnode();
            cnt[cur = nxt] += 1;
        }
    }
    void build() {
        queue<int> q;
        q.push(0);
        while (!q.empty()) {
            int cur = q.front();
            q.pop();
            for (int i = 0; i < CNUM; ++i)
                if (next[cur][i])
                    q.push(insertSAM(cur, i));
        }
    }
}

```

```

vector<int> lc(tot);
for (int i = 1; i < tot; ++i) ++lc[len[i]];
partial_sum(all(lc), lc.begin());
for (int i = 1; i < tot; ++i) lenSorted[--lc[len[i]]] = i;
}
void solve() {
    for (int i = tot - 2; i >= 0; --i)
        cnt[link[lenSorted[i]]] += cnt[lenSorted[i]];
}
};

```

## 5.9 PalindromeTree

```

struct palindromic_tree {
    struct node {
        int next[26], fail, len;
        int cnt, num; // cnt: appear times, num: number of
                        // pal. suf.
        node(int l = 0) : fail(0), len(l), cnt(0), num(0) {
            for (int i = 0; i < 26; ++i) next[i] = 0;
        }
    };
    vector<node> St;
    vector<char> s;
    int last, n;
    palindromic_tree() : St(2), last(1), n(0) {
        St[0].fail = 1, St[1].len = -1, s.emplace_back(-1);
    }
    inline void clear() {
        St.clear(), s.clear(), last = 1, n = 0;
        St.emplace_back(0), St.emplace_back(-1);
        St[0].fail = 1, s.emplace_back(-1);
    }
    inline int get_fail(int x) {
        while (s[n - St[x].len - 1] != s[n])
            x = St[x].fail;
        return x;
    }
    inline void add(int c) {
        s.push_back(c - 'a'), ++n;
        int cur = get_fail(last);
        if (!St[cur].next[c]) {
            int now = St.size();
            St.emplace_back(St[cur].len + 2);
            St[now].fail =
                St[get_fail(St[cur].fail)].next[c];
            St[cur].next[c] = now;
            St[now].num = St[St[now].fail].num + 1;
        }
        last = St[cur].next[c], ++St[last].cnt;
    }
    inline void count() { // counting cnt
        auto i = St.rbegin();
        for (; i != St.rend(); ++i) {
            St[i->fail].cnt += i->cnt;
        }
    }
    inline int size() { // The number of diff. pal.
        return (int)St.size() - 2;
    }
};

```

## 6 Number Theory

### 6.1 Primes

```

12721 13331 14341 75577 123457 222557 556679 999983
1097774749 1076767633 100102021 999997771 1001010013
1000512343 987654361 999991231 999888733 98789101
98777773 999991921 10101333 1010102101 1000000000039
1000000000000037 2305843009213693951 4611686018427387847
9223372036854775783 18446744073709551557

```

### 6.2 ExtGCD

```

// beware of negative numbers!
void extgcd(ll a, ll b, ll c, ll &x, ll &y) {
    if (b == 0) x = c / a, y = 0;
}

```

```

else {
    extgcd(b, a % b, c, y, x);
    y -= x * (a / b);
}
} // |x| <= b/2, |y| <= a/2

```

### 6.3 FloorCeil

```

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

```

### 6.4 FloorSum

```

ll floorsum(ll A, ll B, ll C, ll N) {
    if (A == 0) return (N + 1) * (B / C);
    if (A > C || B > C)
        return (N + 1) * (B / C) +
            N * (N + 1) / 2 * (A / C) +
            floorsum(A % C, B % C, C, N);
    ll M = (A * N + B) / C;
    return N * M - floorsum(C, C - B - 1, A, M - 1);
} // \sum_{i=0}^N floor((ai + b) / m)

```

### 6.5 MillerRabin

```

// n < 4,759,123,141      3 : 2, 7, 61
// n < 1,122,004,669,633  4 : 2, 13, 23, 1662803
// n < 3,474,749,660,383  6 : primes <= 13
// n < 2^64              7 :
// 2, 325, 9375, 28178, 450775, 9780504, 1795265022
bool Miller_Rabin(ll a, ll n) {
    if ((a = a % n) == 0) return 1;
    if (n % 2 == 0) return n == 2;
    ll tmp = (n - 1) / ((n - 1) & (1 - n));
    ll t = __lg(((n - 1) & (1 - n))), x = 1;
    for (; tmp; tmp >>= 1, a = mul(a, a, n))
        if (tmp & 1) x = mul(x, a, n);
    if (x == 1 || x == n - 1) return 1;
    while (--t)
        if ((x = mul(x, x, n)) == n - 1) return 1;
    return 0;
}

```

### 6.6 PollardRho

```

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

```

### 6.7 Fraction

```

struct fraction {
    ll n, d;
    fraction
        (const ll &n=0, const ll &d=1): n(_n), d(_d) {
        ll t = gcd(n, d);
        n /= t, d /= t;
        if (d < 0) n = -n, d = -d;
    }
    fraction operator-(const

```

```

        fraction(-n, d); }
    fraction operator+(const fraction &b) const
    { return fraction(n * b.d + b.n * d, d * b.d); }
    fraction operator-(const fraction &b) const
    { return fraction(n * b.d - b.n * d, d * b.d); }
    fraction operator*(const fraction &b) const
    { return fraction(n * b.n, d * b.d); }
    fraction operator/(const fraction &b) const
    { return fraction(n * b.d, d * b.n); }
    void print() {
        cout << n;
        if (d != 1) cout << "/" << d;
    }
};

```

### 6.8 ChineseRemainder

```

ll solve(ll x1, ll m1, ll x2, ll m2) {
    ll g = gcd(m1, m2);
    if ((x2 - x1) % g) return -1; // no sol
    m1 /= g; m2 /= g;
    ll x, y;
    extgcd(m1, m2, __gcd(m1, m2), x, y);
    ll lcm = m1 * m2 * g;
    ll res = x * (x2 - x1) * m1 + x1;
    // be careful with overflow
    return (res % lcm + lcm) % lcm;
}

```

### 6.9 FactorialMod $p^k$

```

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

```

### 6.10 QuadraticResidue

```

// Berlekamp-Rabin,  $\log^2(p)$ 
ll trial(ll y, ll z, ll m) {
    ll a0 = 1, a1 = 0, b0 = z, b1 = 1, p = (m - 1) / 2;
    while (p) {
        if (p & 1)
            tie(a0, a1) =
                make_pair((a1 * b1 % m * y + a0 * b0) % m,
                    (a0 * b1 + a1 * b0) % m);
            tie(b0, b1) =
                make_pair((b1 * b1 % m * y + b0 * b0) % m,
                    (2 * b0 * b1) % m);
        p >>= 1;
    }
    if (a1) return inv(a1, m);
    return -1;
}
mt19937 rd(49);
ll psqrt(ll y, ll p) {
    if (fpow(y, (p - 1) / 2, p) != 1) return -1;
    for (int i = 0; i < 30; i++) {
        ll z = rd() % p;
        if (z * z % p == y) return z;
        ll x = trial(y, z, p);
        if (x == -1) continue;
        return x;
    }
    return -1;
}

```



## 6.11 MeisselLehmer

```
ll PrimeCount(ll n) { // n ~ 10^13 => < 2s
    if (n <= 1) return 0;
    int v = sqrt(n), s = (v + 1) / 2, pc = 0;
    vector<int> smalls(v + 1), skip(v + 1), roughs(s);
    vector<ll> larges(s);
    for (int i = 2; i <= v; ++i) smalls[i] = (i + 1) / 2;
    for (int i = 0; i < s; ++i) {
        roughs[i] = 2 * i + 1;
        larges[i] = (n / (2 * i + 1) + 1) / 2;
    }
    for (int p = 3; p <= v; ++p) {
        if (smalls[p] > smalls[p - 1]) {
            int q = p * p;
            ++pc;
            if (1LL * q * q > n) break;
            skip[p] = 1;
            for (int i = q; i <= v; i += 2 * p) skip[i] = 1;
            int ns = 0;
            for (int k = 0; k < s; ++k) {
                int i = roughs[k];
                if (skip[i]) continue;
                ll d = 1LL * i * p;
                larges[ns] = larges[k] - (d <= v ? larges[smalls[d] - pc] : smalls[n / d]) + pc;
                roughs[ns++] = i;
            }
            s = ns;
            for (int j = v / p; j >= p; --j) {
                int c =
                    smalls[j] - pc, e = min(j * p + p, v + 1);
                for (int i = j * p; i < e; ++i) smalls[i] -= c;
            }
        }
    }
    for (int k = 1; k < s; ++k) {
        const ll m = n / roughs[k];
        ll t = larges[k] - (pc + k - 1);
        for (int l = 1; l < k; ++l) {
            int p = roughs[l];
            if (1LL * p * p > m) break;
            t -= smalls[m / p] - (pc + l - 1);
        }
        larges[0] -= t;
    }
    return larges[0];
}
```

## 6.12 DiscreteLog

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

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

## 6.13 Theorems

- Cramer's rule

$$\begin{aligned} ax+by &= e \\ cx+dy &= f \end{aligned} \Rightarrow \begin{aligned} x &= \frac{ed-bf}{ad-bc} \\ y &= \frac{af-ec}{ad-bc} \end{aligned}$$

- Vandermonde's Identity

$$C(n+m, k) = \sum_{i=0}^k C(n, i) C(m, k-i)$$

- Kirchhoff's Theorem

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

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

- Tutte's Matrix

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

- Cayley's Formula

- Given a degree sequence  $d_1, d_2, \dots, d_n$  for each labeled vertices, there are  $\frac{(n-2)!}{(d_1-1)!(d_2-1)! \dots (d_n-1)!}$  spanning trees.
- Let  $T_{n,k}$  be the number of labeled forests on  $n$  vertices with  $k$  components, such that vertex  $1, 2, \dots, k$  belong to different components. Then  $T_{n,k} = kn^{n-k-1}$ .

- Erdős-Gallai theorem

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

$$\sum_{i=1}^k d_i \leq k(k-1) + \sum_{i=k+1}^n \min(d_i, k) \text{ holds for every } 1 \leq k \leq n.$$

- Gale-Ryser theorem

A pair of sequences of nonnegative integers  $a_1 \geq \dots \geq a_n$  and  $b_1, \dots, b_n$  is bigraphic if and only if  $\sum_{i=1}^n a_i = \sum_{i=1}^n b_i$  and

$$\sum_{i=1}^k a_i \leq \sum_{i=1}^n \min(b_i, k) \text{ holds for every } 1 \leq k \leq n.$$

- Fulkerson-Chen-Anstee theorem

A sequence  $(a_1, b_1), \dots, (a_n, b_n)$  of nonnegative integer pairs with  $a_1 \geq \dots \geq a_n$  is digraphic if and only if  $\sum_{i=1}^n a_i = \sum_{i=1}^n b_i$

$$\text{and } \sum_{i=1}^k a_i \leq \sum_{i=1}^k \min(b_i, k-1) + \sum_{i=k+1}^n \min(b_i, k) \text{ holds for every } 1 \leq k \leq n.$$

- Pick's theorem

For simple polygon, when points are all integer, we have  $A = \frac{\#\{\text{lattice points in the interior}\} + \#\{\text{lattice points on the boundary}\}}{2} - 1$ .

- Möbius inversion formula

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

- Spherical cap

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

- Lagrange multiplier

- Optimize  $f(x_1, \dots, x_n)$  when  $k$  constraints  $g_i(x_1, \dots, x_n) = 0$ .
- Lagrangian function  $\mathcal{L}(x_1, \dots, x_n, \lambda_1, \dots, \lambda_k) = f(x_1, \dots, x_n) + \sum_{i=1}^k \lambda_i g_i(x_1, \dots, x_n)$ .
- The solution corresponding to the original constrained optimization is always a saddle point of the Lagrangian function.

- Nearest points of two skew lines

- Line 1:  $v_1 = p_1 + t_1 d_1$
- Line 2:  $v_2 = p_2 + t_2 d_2$
- $n = d_1 \times d_2$
- $n_1 = d_1 \times n$
- $n_2 = d_2 \times n$
- $c_1 = p_1 + \frac{(p_2 - p_1) \cdot n_2}{d_1 \cdot n_2} d_1$
- $c_2 = p_2 + \frac{(p_1 - p_2) \cdot n_1}{d_2 \cdot n_1} d_2$

## 6.14 Estimation

### • Estimation

- The number of divisors of  $n$  is at most around 100 for  $n < 5e4$ , 500 for  $n < 1e7$ , 2000 for  $n < 1e10$ , 200000 for  $n < 1e19$ .
- The number of ways of writing  $n$  as a sum of positive integers, disregarding the order of the summands. 1,1,2,3,5,7,11,15,22,30 for  $n = 0 \sim 9$ , 627 for  $n = 20$ ,  $\sim 2e5$  for  $n = 50$ ,  $\sim 2e8$  for  $n = 100$ .
- Total number of partitions of  $n$  distinct elements:  $B(n) = 1, 1, 2, 5, 15, 52, 203, 877, 4140, 21147, 115975, 678570, 4213597, 27644437, 190899322, \dots$

## 6.15 EuclideanAlgorithms

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

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

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

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

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

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

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

## 6.16 Numbers

### • Bernoulli numbers

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

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

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

### • Stirling numbers of the second kind Partitions of $n$ distinct elements into exactly $k$ groups.

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

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

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

### • Pentagonal number theorem

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

### • Catalan numbers

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

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

### • Eulerian numbers

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

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

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

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

## 6.17 GeneratingFunctions

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

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

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

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

### • Special Generating Function

- $(1+x)^n = \sum_{i \geq 0} \binom{n}{i} x^i$
- $\frac{1}{(1-x)^n} = \sum_{i \geq 0} \binom{i+n-1}{n-1} x^i$
- $S_k = \sum_{x=1}^n x^k: S = \sum_{p=0}^{\infty} x^p = \frac{e^x - e^{x(n+1)}}{1 - e^x}$

## 7 Linear Algebra

### 7.1 GaussianElimination

```
#undef M
struct matrix { //m variables, n equations
    int n, m;
    fraction M[N][N + 1], sol[N];
    int solve() { //-1: inconsistent, >= 0: rank
        for (int i = 0; i < n; ++i) {
            int piv = 0;
            while (piv < m && !M[i][piv].n) ++piv;
            if (piv == m) continue;
            for (int j = 0; j < n; ++j) {
                if (i == j) continue;
                fraction tmp = -M[j][piv] / M[i][piv];
                for (int k = 0; k <=
                    m; ++k) M[j][k] = tmp * M[i][k] + M[j][k];
            }
        }
        int rank = 0;
        for (int i = 0; i < n; ++i) {
            int piv = 0;
            while (piv < m && !M[i][piv].n) ++piv;
            if (piv == m && M[i][m].n) return -1;
            else if (piv
                < m) ++rank, sol[piv] = M[i][m] / M[i][piv];
        }
        return rank;
    }
};
```

### 7.2 BerlekampMassey

```
template <typename T>
vector<T> BerlekampMassey(const vector<T> &output) {
    vector<T> d(output.size() + 1), me, he;
    for (int f = 0, i = 1; i <= output.size(); ++i) {
        for (int j = 0; j < me.size(); ++j)
            d[i] += output[i - j - 1] * me[j];
        if ((d[i] -= output[i - 1]) == 0) continue;
        if (me.empty()) {
            me.resize(f = i);
            continue;
        }
        vector<T> o(i - f - 1);
        T k = -d[i] / d[f];
        o.emplace_back(-k);
        for (T x : he) o.emplace_back(x * k);
        o.resize(max(o.size(), me.size()));
        for (int j = 0; j < me.size(); ++j) o[j] += me[j];
        if (i - f + (int
            )he.size()) >= (int)me.size()) he = me, f = i;
        me = o;
    }
    return me;
}
```

## 7.3 Simplex

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

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

```
// using N + 2M variables
const int mxM = 25;
const int mxN = 25 + 2 * mxM;
struct simplex {
    const double inf = 1 / .0, eps = 1e-9;
    int n, m, k, var[mxN], inv[mxN], art[mxN];
    double A[mxM][mxN], B[mxM], x[mxN];
    void init(int _n) { n = _n, m = 0; }
    void equation(vector<double> a, double b) {
        for (int i = 0; i < n; i++) A[m][i] = a[i];
        B[m] = b, var[m] = n + m, ++m;
    }
    void pivot(int r, int c, double bx) {
        for (int i = 0; i <= m + 1; i++)
            if (i != r && abs(A[i][c]) > eps) {
                x[var[i]] -= bx * A[i][c] / A[r][c];
                double f = A[i][c] / A[r][c];
                for (int j = 0; j <= n + m + k; j++)
                    A[i][j] -= A[r][j] * f;
                B[i] -= B[r] * f;
            }
    }
    double phase(int p) {
        while (true) {
            int in = min_element(
                A[m + p], A[m + p] + n + m + k + 1) -
                A[m + p];
            if (A[m + p][in] >= -eps) break;
            double bx = inf;
            int piv = -1;
            for (int i = 0; i < m; i++)
                if (A[i][in] > eps && B[i] / A[i][in] <= bx)
                    piv = i, bx = B[i] / A[i][in];
            if (piv == -1) return inf;
            int out = var[piv];
            pivot(piv, in, bx);
            x[out] = 0, x[in] = bx, var[piv] = in;
        }
        return x[n + m];
    }
    double solve(vector<double> c) {
        auto invert = [&](int r) {
            for (int j = 0; j <= n + m; j++) A[r][j] *= -1;
            B[r] *= -1;
        };
        k = 1;
        for (int i = 0; i < n; i++) A[m][i] = -c[i];
        fill(A[m + 1], A[m + 1] + mxN, 0.0);
        for (int i = 0; i <= m + 1; i++)
            fill(A[i] + n, A[i] + n + m + 2, 0.0),
            var[i] = n + i, A[i][n + i] = 1;
        for (int i = 0; i < m; i++) {
            if (B[i] < 0) {
                ++k;
                for (int j = 0; j <= n + m; j++)
                    A[m + 1][j] += A[i][j];
                invert(i);
                var[i] = n + m + k, A[i][var[i]] = 1,
                art[var[i]] = n + i;
            }
            x[var[i]] = B[i];
        }
    }
}
```

```
phase(1);
if (*max_element(
    x + (n + m + 2), x + (n + m + k + 1)) > eps)
    return .0 / .0;
for (int i = 0; i <= m; i++)
    if (var[i] > n + m)
        var[i] = art[var[i]], invert(i);
k = 0;
return phase(0);
}
} lp;
```

## 8 Polynomials

### 8.1 NTT (FFT)

```
// 9223372036737335297, 3
#define base ll // complex<double>
#define N 524288
// const double PI = acos(-1);
const ll mod = 998244353, g = 3;
base omega[4 * N], omega_[4 * N];
int rev[4 * N];

ll fpow(ll b, ll p);

ll inverse(ll a) { return fpow(a, mod - 2); }

void calcW(int n) {
    ll r = fpow(g, (mod - 1) / n), invr = inverse(r);
    omega[0] = omega_[0] = 1;
    for (int i = 1; i < n; i++) {
        omega[i] = omega[i - 1] * r % mod;
        omega_[i] = omega_[i - 1] * invr % mod;
    }
    // double arg = 2.0 * PI / n;
    // for (int i = 0; i < n; i++)
    // {
    //     omega[i] = base(cos(i * arg), sin(i * arg));
    //     omega_[i] = base(cos(-i * arg), sin(-i * arg));
    // }
}

void calcrev(int n) {
    int k = __lg(n);
    for (int i = 0; i < n; i++) rev[i] = 0;
    for (int i = 0; i < n; i++)
        for (int j = 0; j < k; j++)
            if (i & (1 << j)) rev[i] ^= 1 << (k - j - 1);
}

vector<base> NTT(vector<base> poly, bool inv) {
    base *w = (inv ? omega_ : omega);
    int n = poly.size();
    for (int i = 0; i < n; i++)
        if (rev[i] > i) swap(poly[i], poly[rev[i]]);

    for (int len = 1; len < n; len <= 1) {
        int arg = n / len / 2;
        for (int i = 0; i < n; i += 2 * len)
            for (int j = 0; j < len; j++) {
                base odd =
                    w[j * arg] * poly[i + j + len] % mod;
                poly[i + j + len] =
                    (poly[i + j] - odd + mod) % mod;
                poly[i + j] = (poly[i + j] + odd) % mod;
            }
    }
    if (inv)
        for (auto &a : poly) a = a * inverse(n) % mod;
    return poly;
}

vector<base> mul(vector<base> f, vector<base> g) {
    int sz = 1 << (__lg(f.size() + g.size() - 1) + 1);
    f.resize(sz), g.resize(sz);
```

```

calcrev(sz);
calcW(sz);
f = NTT(f, 0), g = NTT(g, 0);
for (int i = 0; i < sz; i++)
    f[i] = f[i] * g[i] % mod;
return NTT(f, 1);
}

```

## 8.2 FHWT

```

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

```

## 8.3 PolynomialOperations

```

#define poly vector<ll>
poly inv(poly A) {
    A.resize(1 << (lg(A.size()) - 1) + 1));
    poly B = {inverse(A[0])};
    for (int n = 1; n < A.size(); n += n) {
        poly pA(A.begin(), A.begin() + 2 * n);
        calcrev(4 * n);
        calcW(4 * n);
        pA.resize(4 * n);
        B.resize(4 * n);
        pA = NTT(pA, 0);
        B = NTT(B, 0);
        for (int i = 0; i < 4 * n; i++)
            B[i] =
                ((B[i] * 2 - pA[i] * B[i] % mod * B[i]) % mod +
                 mod) %
                mod;
        B = NTT(B, 1);
        B.resize(2 * n);
    }
    return B;
}

pair<poly, poly> div(poly A, poly B) {
    if (A.size() < B.size()) return make_pair(poly(), A);
    int n = A.size(), m = B.size();
    poly revA = A, invrevB = B;
    reverse(revA.begin(), revA.end());
    reverse(invrevB.begin(), invrevB.end());
    revA.resize(n - m + 1);
    invrevB.resize(n - m + 1);
    invrevB = inv(invrevB);
}

```

```

poly Q = mul(revA, invrevB);
Q.resize(n - m + 1);
reverse(Q.begin(), Q.end());
poly R = mul(Q, B);
R.resize(m - 1);
for (int i = 0; i < m - 1; i++)
    R[i] = (A[i] - R[i] + mod) % mod;
return make_pair(Q, R);
}

```

```

ll fast_kitamasu(ll k, poly A, poly C) {
    int n = A.size();
    C.emplace_back(mod - 1);
    poly Q, R = {0, 1}, F = {1};
    R = div(R, C);
    while (k) {
        if (k & 1) F = div(mul(F, R), C);
        R = div(mul(R, R), C);
        k >>= 1;
    }
    ll ans = 0;
    for (int i = 0; i < F.size(); i++)
        ans = (ans + A[i] * F[i]) % mod;
    return ans;
}

vector<ll> fpow(vector<ll> f, ll p, ll m) {
    int b = 0;
    while (b < f.size() && f[b] == 0) b++;
    f = vector<ll>(f.begin() + b, f.end());
    int n = f.size();
    f.emplace_back(0);
    vector<ll> q(min(m, b * p), 0);
    q.emplace_back(fpow(f[0], p));
    for (int k = 0; q.size() < m; k++) {
        ll res = 0;
        for (int i = 0; i < min(n, k + 1); i++)
            res = (res +
                    p * (i + 1) % mod * f[i + 1] % mod *
                    q[k - i + b * p]) %
                    mod;
        for (int i = 1; i < min(n, k + 1); i++)
            res = (res -
                    f[i] * (k - i + 1) % mod *
                    q[k - i + 1 + b * p]) %
                    mod;
        res = (res < 0 ? res + mod : res) *
            inv(f[0] * (k + 1) % mod) % mod;
        q.emplace_back(res);
    }
    return q;
}

```

## 8.4 NewtonMethod+MiscGF

Given  $F(x)$  where

$$F(x) = \sum_{i=0}^{\infty} \alpha_i (x - \beta)^i$$

for  $\beta$  being some constant. Polynomial  $P$  such that  $F(P) = 0$  can be found iteratively. Denote by  $Q_k$  the polynomial such that  $F(Q_k) = 0 \pmod{x^{2^k}}$ , then

$$Q_{k+1} = Q_k - \frac{F(Q_k)}{F'(Q_k)} \pmod{x^{2^{k+1}}}$$

- $A^{-1}$ :  $B_{k+1} = B_k(2 - AB_k) \pmod{x^{2^{k+1}}}$
- $\ln A$ :  $(\ln A)' = \frac{A'}{A}$
- $\exp A$ :  $B_{k+1} = B_k(1 + A - \ln B_k) \pmod{x^{2^{k+1}}}$
- $\sqrt{A}$ :  $B_{k+1} = \frac{1}{2}(B_k + AB_k^{-1}) \pmod{x^{2^{k+1}}}$

## 9 Geometry

### 9.1 Basic

```

typedef pair<pdd, pdd> Line;
struct Cir { pdd O; double R; };
const double eps = 1e-8;

```

```

pdd operator+(pdd a, pdd b)
{ return pdd(a.F + b.S, a.S + b.S); }
pdd operator-(pdd a, pdd b)
{ return pdd(a.F - b.S, a.S - b.S); }
pdd operator*(pdd a, double b)
{ return pdd(a.F * b, a.S * b); }
pdd operator/(pdd a, double b)
{ return pdd(a.F / b, a.S / b); }
double dot(pdd a, pdd b)
{ return a.F * b.F + a.S * b.S; }
double cross(pdd a, pdd b)
{ return a.F * b.S - a.S * b.F; }
double abs2(pdd a)
{ return dot(a, a); }
double abs(pdd a)
{ return sqrt(dot(a, a)); }
int sign(double a)
{ return fabs(a) < eps ? 0 : a > 0 ? 1 : -1; }
int ori(pdd a, pdd b, pdd c)
{ return sign(cross(b - a, c - a)); }
bool collinearity(pdd p1, pdd p2, pdd p3)
{ return sign(cross(p1 - p3, p2 - p3)) == 0; }
bool btw(pdd p1, pdd p2, pdd p3) {
    if (!collinearity(p1, p2, p3)) return 0;
    return sign(dot(p1 - p3, p2 - p3)) <= 0;
}
bool seg_intersect(pdd p1, pdd p2, pdd p3, pdd p4) {
    int a123 = ori(p1, p2, p3);
    int a124 = ori(p1, p2, p4);
    int a341 = ori(p3, p4, p1);
    int a342 = ori(p3, p4, p2);
    if (a123 == 0 && a124 == 0)
        return btw(p1, p2, p3) || btw(p1, p2, p4) ||
            btw(p3, p4, p1) || btw(p3, p4, p2);
    return a123 * a124 <= 0 && a341 * a342 <= 0;
}
pdd intersect(pdd p1, pdd p2, pdd p3, pdd p4) {
    double a123 = cross(p2 - p1, p3 - p1);
    double a124 = cross(p2 - p1, p4 - p1);
    return (p4
        * a123 - p3 * a124) / (a123 - a124); // C^3 / C^2
}
pdd perp(pdd p1)
{ return pdd(-p1.S, p1.F); }
pdd projection(pdd p1, pdd p2, pdd p3)
{ return p1 + (
    p2 - p1) * dot(p3 - p1, p2 - p1) / abs2(p2 - p1); }
pdd reflection(pdd p1, pdd p2, pdd p3)
{ return p3 + perp(p2 - p1
    ) * cross(p3 - p1, p2 - p1) / abs2(p2 - p1) * 2; }
pdd linearTransformation
    (pdd p0, pdd p1, pdd q0, pdd q1, pdd r) {
    pdd dp = p1 - p0
        , dq = q1 - q0, num(cross(dp, dq), dot(dp, dq));
    return q0 + pdd(
        cross(r - p0, num), dot(r - p0, num)) / abs2(dp);
} // from line p0--p1 to q0--q1, apply to r

```

## 9.2 ConvexHull

```

void hull(vector<pll> &dots) { // n=1 => ans = {}
    sort(dots.begin(), dots.end());
    vector<pll> ans(1, dots[0]);
    for (int ct = 0; ct < 2; ++ct, reverse(all(dots)))
        for (int i = 1, t = ans.size()
            ; i < dots.size(); ans.emplace_back(dots[i++]))
            while (ans.size() > t &&
                ori(ans.end()[-2], ans.back(), dots[i]) <= 0)
                ans.pop_back();
    ans.pop_back(), ans.swap(dots);
}

```

## 9.3 SortByAngle

```

int cmp(pll a, pll b, bool same = true) {
#define is_neg(k) (
    sign(k.S) < 0 || (sign(k.S) == 0 && sign(k.F) < 0))

```

```

int A = is_neg(a), B = is_neg(b);
if (A != B)
    return A < B;
if (sign(cross(a, b)) == 0)
    return same ? abs2(a) < abs2(b) : -1;
return sign(cross(a, b)) > 0;
}

```

## 9.4 DisPointSegment

```

double PointSegDist(pdd q0, pdd q1, pdd p) {
    if (sign(abs(q0 - q1)) == 0) return abs(q0 - p);
    if (sign(dot(q1 - q0,
        p - q0)) >= 0 && sign(dot(q0 - q1, p - q1)) >= 0)
        return fabs(cross(q1 - q0, p - q0) / abs(q0 - q1));
    return min(abs(p - q0), abs(p - q1));
}

```

## 9.5 PointInCircle

```

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

```

## 9.6 PointInConvex

```

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

```

## 9.7 PointTangentConvex

```

/* The point should be strictly out of hull
   return arbitrary point on the tangent line */
/* bool pred(int a, int b);
f(0) ~ f(n - 1) is a cyclic-shift U-function
return idx s.t. pred(x, idx) is false forall x*/
int cyc_tsearch(int n, auto pred) {
    if (n == 1) return 0;
    int l = 0, r = n; bool rv = pred(1, 0);
    while (r - l > 1) {
        int m = (l + r) / 2;
        if (pred(0, m) ? rv : pred(m, (m + 1) % n)) r = m;
        else l = m;
    }
    return pred(l, r % n) ? l : r % n;
}
pii get_tangent(vector<pll> &C, pll p) {
    auto gao = [&](int s) {
        return cyc_tsearch((int)C.size(), [&](int x, int y)
            { return ori(p, C[x], C[y]) == s; });
    };
    return pii(gao(1), gao(-1));
} // return (a, b), ori(p, C[a], C[b]) >= 0

```



## 9.8 CircTangentCirc

```
vector<Line>
> go( const Cir& c1 , const Cir& c2 , int sign1 ){
// sign1 = 1 for outer tang, -1 for inter tang
vector<Line> ret;
double d_sq = abs2(c1.0 - c2.0);
if (sign(d_sq) == 0) return ret;
double d = sqrt(d_sq);
pdd v = (c2.0 - c1.0) / 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) {
    pdd n = pdd(v.F * c - sign2 * h * v.S,
        v.S * c + sign2 * h * v.F);
    pdd p1 = c1.0 + n * c1.R;
    pdd p2 = c2.0 + n * (c2.R * sign1);
    if (sign(p1.F - p2.F) == 0 and
        sign(p1.S - p2.S) == 0)
        p2 = p1 + perp(c2.0 - c1.0);
    ret.emplace_back(Line(p1, p2));
}
return ret;
}
```

## 9.9 LineCircleIntersect

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

## 9.10 LineConvexIntersect

```
int TangentDir(vector<pll> &C, pll dir) {
    return cyc_tsearch((int)C.size(), [&](int a, int b) {
        return cross(dir, C[a]) > cross(dir, C[b]);
    });
}
#define cmpL(i) sign(cross(C[i] - a, b - a))
pii lineHull(pll a, pll b, vector<pll> &C) {
    int A = TangentDir(C, a - b);
    int B = TangentDir(C, b - a);
    int n = (int)C.size();
    if (cmpL(A) < 0 || cmpL(B) > 0)
        return pii(-1, -1); // no collision
    auto gao = [&](int l, int r) {
        for (int t = l; (l + 1) % n != r; ) {
            int m = ((l + r + (l < r ? 0 : n)) / 2) % n;
            (cmpL(m) == cmpL(t) ? l : r) = m;
        }
        return (l + !cmpL(r)) % n;
    };
    pii res = pii(gao(B, A), gao(A, B)); // (i, j)
    if (res.F == res.S) // touching the corner i
        return pii(res.F, -1);
    if (!
        cmpL(res.F) && !cmpL(res.S)) // along side i, i+1
        switch ((res.F - res.S + n + 1) % n) {
            case 0: return pii(res.F, res.F);
            case 2: return pii(res.S, res.S);
        }
    /* crossing sides (i, i+1) and (j, j+1)
    crossing corner i is treated as side (i, i+1)
    returned
    in the same order as the line hits the convex */
    return res;
} // convex cut: (r, l)
```

## 9.11 CircIntersectCirc

```
bool CCinter(Cir &a, Cir &b, pdd &p1, pdd &p2) {
    pdd o1 = a.0, o2 = b.0;
    double r1 =
        a.R, r2 = b.R, d2 = abs2(o1 - o2), d = sqrt(d2);
    if (d < max
        (r1, r2) - min(r1, r2) || d > r1 + r2) return 0;
    pdd u = (o1 + o2) * 0.5
        + (o1 - o2) * ((r2 * r2 - r1 * r1) / (2 * d2));
    double A = sqrt((r1 + r2 + d) *
        (r1 - r2 + d) * (r1 + r2 - d) * (-r1 + r2 + d));
    pdd v
        = pdd(o1.S - o2.S, -o1.F + o2.F) * A / (2 * d2);
    p1 = u + v, p2 = u - v;
    return 1;
}
```

## 9.12 PolyIntersectCirc

```
// Divides into multiple triangle, and sum up
const double PI = acos(-1);
double _area(pdd pa, pdd pb, double r) {
    if (abs(pa) < abs(pb)) swap(pa, pb);
    if (abs(pb) < eps) return 0;
    double S, h, theta;
    double a = abs(pb), b = abs(pa), c = abs(pb - pa);
    double cosB = dot(pb, pb - pa) / a / c,
        B = acos(cosB);
    double cosC = dot(pa, pb) / a / b, C = acos(cosC);
    if (a > r) {
        S = (C / 2) * r * r;
        h = a * b * sin(C) / c;
        if (h < r && B < PI / 2)
            S -= (acos(h / r) * r * r -
                h * sqrt(r * r - h * h));
    } else if (b > r) {
        theta = PI - B - asin(sin(B) / r * a);
        S = .5 * a * r * sin(theta) +
            (C - theta) / 2 * r * r;
    } else S = .5 * sin(C) * a * b;
    return S;
}
double area_poly_circle(const vector<pdd> poly,
    const pdd &o, const double r) {
    double S = 0;
    for (int i = 0; i < (int)poly.size(); ++i)
        S += _area(poly[i] - o,
            poly[(i + 1) % (int)poly.size()] - o, r) *
            ori(
                0, poly[i], poly[(i + 1) % (int)poly.size()]);
    return fabs(S);
}
```

## 9.13 MinkowskiSum

```
vector<pll> Minkowski
    (vector<pll> A, vector<pll> B) { // |A|, |B| >= 3
    hull(A), hull(B);
    vector<pll> C(1, A[0] + B[0]), s1, s2;
    for (int i = 0; i < A.size(); ++i)
        s1.emplace_back(A[(i + 1) % A.size()] - A[i]);
    for (int i = 0; i < B.size(); ++i)
        s2.emplace_back(B[(i + 1) % B.size()] - B[i]);
    for (int i = 0, j = 0; i < A.size() || j < B.size(); )
        if (j >= B.size()
            || (i < A.size() && cross(s1[i], s2[j]) >= 0))
            C.emplace_back(B[j % B.size()] + A[i++]);
        else
            C.emplace_back(A[i % A.size()] + B[j++]);
    return hull(C), C;
}
```

## 9.14 MinMaxEnclosingRect

```
const double INF = 1e18, qi = acos(-1) / 2 * 3;
pdd solve(vector<pll> &dots) {
#define diff(u, v) (dots[u] - dots[v])
```



```

#define vec(v) (dots[v] - dots[i])
hull(dots);
double Max = 0, Min = INF, deg;
int n = (int)dots.size();
dots.emplace_back(dots[0]);
for (int i = 0, u = 1, r = 1, l = 1; i < n; ++i) {
    pll nw = vec(i + 1);
    while (cross(nw, vec(u + 1)) > cross(nw, vec(u)))
        u = (u + 1) % n;
    while (dot(nw, vec(r + 1)) > dot(nw, vec(r)))
        r = (r + 1) % n;
    if (!i) l = (r + 1) % n;
    while (dot(nw, vec(l + 1)) < dot(nw, vec(l)))
        l = (l + 1) % n;
    Min = min(Min, (double)(dot(nw, vec(r)) - dot
        (nw, vec(l))) * cross(nw, vec(u)) / abs2(nw));
    deg = acos(dot(diff(r
        , l), vec(u)) / abs(diff(r, l)) / abs(vec(u)));
    deg = (qi - deg) / 2;
    Max = max(Max, abs(diff
        (r, l)) * abs(vec(u)) * sin(deg) * sin(deg));
}
return pdd(Min, Max);
}

```

## 9.15 MinEnclosingCircle

```

pdd Minimum_Enclosing_Circle
(vector<pdd> dots, double &r) {
    pdd cent;
    random_shuffle(all(dots));
    cent = dots[0], r = 0;
    for (int i = 1; i < (int)dots.size(); ++i)
        if (abs(dots[i] - cent) > r) {
            cent = dots[i], r = 0;
            for (int j = 0; j < i; ++j)
                if (abs(dots[j] - cent) > r) {
                    cent = (dots[i] + dots[j]) / 2;
                    r = abs(dots[i] - cent);
                    for (int k = 0; k < j; ++k)
                        if (abs(dots[k] - cent) > r)
                            cent = excenter
                                (dots[i], dots[j], dots[k], r);
                }
        }
    return cent;
}

```

## 9.16 CircleCover

```

const int N = 1021;
struct CircleCover {
    int C;
    Cir c[N];
    bool g[N][N], overlap[N][N];
    // Area[i] : area covered by at least i circles
    double Area[ N ];
    void init(int _C){ C = _C;}
    struct Teve {
        pdd p; double ang; int add;
        Teve() {}
        Teve(pdd _a
            , double _b, int _c):p(_a), ang(_b), add(_c){}
        bool operator<(const Teve &a)const
        {return ang < a.ang;}
    }eve[N * 2];
    // strict: x = 0, otherwise x = -1
    bool disjuct(Cir &a, Cir &b, int x)
    {return sign(abs(a.o - b.o) - a.R - b.R) > x;}
    bool contain(Cir &a, Cir &b, int x)
    {return sign(a.R - b.R - abs(a.o - b.o)) > x;}
    bool contain(int i, int j) {
        /* c[j] is non-strictly in c[i]. */
        return (sign
            (c[i].R - c[j].R) > 0 || (sign(c[i].R - c[j].
            R) == 0 && i < j)) && contain(c[i], c[j], -1);
    }
}

```

```

void solve(){
    fill_n(Area, C + 2, 0);
    for(int i = 0; i < C; ++i)
        for(int j = 0; j < C; ++j)
            overlap[i][j] = contain(i, j);
    for(int i = 0; i < C; ++i)
        for(int j = 0; j < C; ++j)
            g[i][j] = !(overlap[i][j] || overlap[j][i] ||
                disjuct(c[i], c[j], -1));
    for(int i = 0; i < C; ++i){
        int E = 0, cnt = 1;
        for(int j = 0; j < C; ++j)
            if(j != i && overlap[j][i])
                ++cnt;
        for(int j = 0; j < C; ++j)
            if(i != j && g[i][j]) {
                pdd aa, bb;
                CCinter(c[i], c[j], aa, bb);
                double A =
                    atan2(aa.Y - c[i].O.Y, aa.X - c[i].O.X);
                double B =
                    atan2(bb.Y - c[i].O.Y, bb.X - c[i].O.X);
                eve[E++] = Teve
                    (bb, B, 1), eve[E++] = Teve(aa, A, -1);
                if(B > A) ++cnt;
            }
        if(E == 0) Area[cnt] += pi * c[i].R * c[i].R;
        else{
            sort(eve, eve + E);
            eve[E] = eve[0];
            for(int j = 0; j < E; ++j){
                cnt += eve[j].add;
                Area[cnt]
                    += cross(eve[j].p, eve[j + 1].p) * .5;
                double theta = eve[j + 1].ang - eve[j].ang;
                if (theta < 0) theta += 2. * pi;
                Area[cnt] += (theta
                    - sin(theta)) * c[i].R * c[i].R * .5;
            }
        }
    }
}

```

## 9.17 LineCmp

```

using Line = pair<pll, pll>;
struct lineCmp {
    bool operator()(Line l1, Line l2) const {
        int X =
            (max(l1.F.F, l2.F.F) + min(l1.S.F, l2.S.F)) / 2;
        ll p1 =
            (X - l1.F.F) * l1.S.S + (l1.S.F - X) * l1.F.S,
            p2 =
            (X - l2.F.F) * l2.S.S + (l2.S.F - X) * l2.F.S,
            q1 = (l1.S.F - l1.F.F), q2 = (l2.S.F - l2.F.F);
        if (q1 == 0) p1 = l1.F.S + l1.S.S, q1 = 2;
        if (q2 == 0) p2 = l2.F.S + l2.S.S, q2 = 2;
        if (l1.F == l2.F || l2.F == l2.S) l1 = l2;
        return make_tuple((__int128)(p1 * q2), l1) <
            make_tuple((__int128)(p2 * q1), l2);
    }
};

```

## 9.18 Trapezoidalization

```

struct SweepLine {
    struct cmp {
        cmp(const SweepLine &swp): swp(swp) {}
        bool operator()(int a, int b) const {
            if (abs(swp.get_y(a) - swp.get_y(b)) <= swp.eps)
                return swp.slope_cmp(a, b);
            return swp.get_y(a) + swp.eps < swp.get_y(b);
        }
    } _cmp;
    T curTime, eps, curQ;
}

```

```

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

```

```

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

```

## 9.19 TriangleHearts

```

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

```

## 9.20 HalfPlaneIntersect

```

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

```

```

    continue;
    while ((int)dq.size() >= 2
        && !isin(p, dq[(int)dq.size() - 2], dq.back()))
        dq.pop_back();
    while
        ((int)dq.size() >= 2 && !isin(p, dq[0], dq[1]))
        dq.pop_front();
    dq.emplace_back(p);
}
while ((int)dq.size() >= 3 &&
    !isin(dq[0], dq[(int)dq.size() - 2], dq.back()))
    dq.pop_back();
while ((int)
    dq.size() >= 3 && !isin(dq.back(), dq[0], dq[1]))
    dq.pop_front();
return vector<Line>(all(dq));
}

```

## 9.21 RotatingSweepLine

```

void rotatingSweepLine(vector<pii> &ps) {
    int n = (int)ps.size(), m = 0;
    vector<int> id(n), pos(n);
    vector<pii> line(n * (n - 1));
    for (int i = 0; i < n; ++i)
        for (int j = 0; j < n; ++j)
            if (i != j) line[m++] = pii(i, j);
    sort(all(line), [&](pii a, pii b) {
        return cmp(ps[a.S] - ps[a.F], ps[b.S] - ps[b.F]);
    }); // cmp(): polar angle compare
    iota(all(id), 0);
    sort(all(id), [&](int a, int b) {
        if (ps[a].S != ps[b].S) return ps[a].S < ps[b].S;
        return ps[a] < ps[b];
    }); // initial order, since (1, 0) is the smallest
    for (int i = 0; i < n; ++i) pos[id[i]] = i;
    for (int i = 0; i < m; ++i) {
        auto l = line[i];
        // do something
        tie(pos[l.F], pos[l.S], id[pos[l.F]], id[pos[l.S]
            ]) = make_tuple(pos[l.S], pos[l.F], l.S, l.F);
    }
}

```

## 9.22 DelaunayTriangulation

```

/* Delaunay Triangulation:
Given a sets of points on 2D plane, find a
triangulation such that no points will strictly
inside circumcircle of any triangle.
find : return a triangle contain given point
add_point : add a point into triangulation
A Triangle is in triangulation iff. its has_chd is 0.
Region of triangle u: iterate each u.edge[i].tri,
each points are u.p[(i+1)%3], u.p[(i+2)%3]
Voronoi diagram: for each triangle in triangulation,
the bisector of all its edges will split the region.
nearest point will belong to the triangle containing it
*/
const
    ll inf = MAXC * MAXC * 100; // lower_bound unknown
struct Tri;
struct Edge {
    Tri* tri; int side;
    Edge(): tri(0), side(0){}
    Edge(Tri* _tri, int _side): tri(_tri), side(_side){}
};
struct Tri {
    pll p[3];
    Edge edge[3];
    Tri* chd[3];
    Tri() {}
    Tri(const pll& p0, const pll& p1, const pll& p2) {
        p[0] = p0; p[1] = p1; p[2] = p2;
        chd[0] = chd[1] = chd[2] = 0;
    }
    bool has_chd() const { return chd[0] != 0; }
    int num_chd() const {

```

```

        return !!chd[0] + !!chd[1] + !!chd[2];
    }
    bool contains(pll const& q) const {
        for (int i = 0; i < 3; ++i)
            if (ori(p[i], p[(i + 1) % 3], q) < 0)
                return 0;
        return 1;
    }
} pool[N * 10], *tris;
void edge(Edge a, Edge b) {
    if(a.tri) a.tri->edge[a.side] = b;
    if(b.tri) b.tri->edge[b.side] = a;
}
struct Trig { // Triangulation
    Trig() {
        the_root
            = // Tri should at least contain all points
            new(tris++) Tri(pll(-inf, -inf),
                pll(inf + inf, -inf), pll(-inf, inf + inf));
    }
    Tri* find(pll p) { return find(the_root, p); }
    void add_point(const
        pll &p) { add_point(find(the_root, p), p); }
    Tri* the_root;
    static Tri* find(Tri* root, const pll &p) {
        while (1) {
            if (!root->has_chd())
                return root;
            for (int i = 0; i < 3 && root->chd[i]; ++i)
                if (root->chd[i]->contains(p)) {
                    root = root->chd[i];
                    break;
                }
        }
        assert(0); // "point not found"
    }
    void add_point(Tri* root, pll const& p) {
        Tri* t[3];
        /* split it into three triangles */
        for (int i = 0; i < 3; ++i)
            t[i] = new(tris
                ++) Tri(root->p[i], root->p[(i + 1) % 3], p);
        for (int i = 0; i < 3; ++i)
            edge(Edge(t[i], 0), Edge(t[(i + 1) % 3], 1));
        for (int i = 0; i < 3; ++i)
            edge(Edge(t[i], 2), root->edge[(i + 2) % 3]);
        for (int i = 0; i < 3; ++i)
            root->chd[i] = t[i];
        for (int i = 0; i < 3; ++i)
            flip(t[i], 2);
    }
    void flip(Tri* tri, int pi) {
        Tri* trj = tri->edge[pi].tri;
        int pj = tri->edge[pi].side;
        if (!trj) return;
        if (!in_cc(tri->p
            [0], tri->p[1], tri->p[2], trj->p[pj])) return;
        /* flip edge between tri, trj */
        Tri* trk = new(tris++) Tri
            (tri->p[(pi + 1) % 3], trj->p[pj], tri->p[pi]);
        Tri* trl = new(tris++) Tri
            (trj->p[(pj + 1) % 3], tri->p[pi], trj->p[pj]);
        edge(Edge(trk, 0), Edge(trl, 0));
        edge(Edge(trk, 1), tri->edge[(pi + 2) % 3]);
        edge(Edge(trk, 2), trj->edge[(pj + 1) % 3]);
        edge(Edge(trl, 1), trj->edge[(pj + 2) % 3]);
        edge(Edge(trl, 2), tri->edge[(pi + 1) % 3]);
        tri->chd
            [0] = trk; tri->chd[1] = trl; tri->chd[2] = 0;
        trj->chd
            [0] = trk; trj->chd[1] = trl; trj->chd[2] = 0;
        flip(trk, 1); flip(trk, 2);
        flip(trl, 1); flip(trl, 2);
    }
};
vector<Tri*> triang; // vector of all triangle

```

```

set<Tri*> vst;
void go(Tri* now) { // store all tri into triang
    if (vst.find(now) != vst.end())
        return;
    vst.insert(now);
    if (!now->has_chd())
        return triang.emplace_back(now);
    for (int i = 0; i < now->num_chd(); ++i)
        go(now->chd[i]);
}
void build(int n, pll* ps) { // build triangulation
    tris = pool; triang.clear(); vst.clear();
    random_shuffle(ps, ps + n);
    Trig tri; // the triangulation structure
    for (int i = 0; i < n; ++i)
        tri.add_point(ps[i]);
    go(tri.the_root);
}

```

## 9.23 VoronoiDiagram

```

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

```

## 10 Misc

### 10.1 MoAlgoWithModify

```

/*
Mo's Algorithm With modification
Block:  $N^{\frac{2}{3}}$ , Complexity:  $N^{\frac{5}{3}}$ 
*/
struct Query {
    int L, R, LBid, RBid, T;
    Query(int l, int r, int t):
        L(l), R(r), LBid(l / blk), RBid(r / blk), T(t) {}
    bool operator<(const Query &q) const {
        if (LBid != q.LBid) return LBid < q.LBid;
        if (RBid != q.RBid) return RBid < q.RBid;
        return T < q.T;
    }
};
void solve(vector<Query> query) {
    sort(ALL(query));
    int L=0, R=0, T=-1;
    for (auto q : query) {
        while (T < q.T) addTime(L, R, ++T); // TODO
        while (T > q.T) subTime(L, R, T--); // TODO
        while (R < q.R) add(arr[++R]); // TODO
        while (L > q.L) add(arr[--L]); // TODO
        while (R > q.R) sub(arr[R--]); // TODO
        while (L < q.L) sub(arr[L++]); // TODO
        // answer query
    }
}

```

### 10.2 MoAlgoOnTree

```

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

```

```

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

```

### 10.3 MoAlgoAdvanced

- Mo's Algorithm With Addition Only
  - Sort queries same as the normal Mo's algorithm.
  - For each query  $[l, r]$ :
  - If  $l/blk = r/blk$ , brute-force.
  - If  $l/blk \neq r/blk$ , initialize  $curL := (l/blk + 1) \cdot blk$ ,  $curR := curL - 1$
  - If  $r > curR$ , increase  $curR$
  - decrease  $curL$  to fit  $l$ , and then undo after answering
- Mo's Algorithm With Offline Second Time
  - Require: Changing answer  $\equiv$  adding  $f([l, r], r+1)$ .
  - Require:  $f([l, r], r+1) = f([l, r], r) + f([l, l], r+1)$ .
  - Part1: Answer all  $f([l, r], r+1)$  first.
  - Part2: Store  $curR \rightarrow R$  for  $curL$  (reduce the space to  $O(N)$ ), and then answer them by the second offline algorithm.
  - Note: You must do the above symmetrically for the left boundaries.

### 10.4 HilbertCurve

```

ll hilbert(int n, int x, int y) {
    ll res = 0;
    for (int s = n / 2; s; s >= 1) {
        int rx = (x & s) > 0;
        int ry = (y & s) > 0;
        res += s * 1ll * s * ((3 * rx) ^ ry);
        if (ry == 0) {
            if (rx == 1) x = s - 1 - x, y = s - 1 - y;
            swap(x, y);
        }
    }
    return res;
} // n = 2^k

```

### 10.5 SternBrocotTree

- Construction: Root  $\frac{1}{1}$ , left/right neighbor  $\frac{0}{1}, \frac{1}{0}$ , each node is sum of last left/right neighbor:  $\frac{a}{b}, \frac{c}{d} \rightarrow \frac{a+c}{b+d}$
- Property: Adjacent (mid-order DFS)  $\frac{a}{b}, \frac{c}{d} \Rightarrow bc - ad = 1$ .
- Search known  $\frac{p}{q}$ : keep L-R alternative. Each step can be calculated in  $O(1) \Rightarrow$  total  $O(\log C)$ .
- Search unknown  $\frac{p}{q}$ : keep L-R alternative. Each step can be calculated in  $O(\log C)$  checks  $\Rightarrow$  total  $O(\log^2 C)$  checks.

## 10.6 ALLLCS

```
void all_lcs(string s, string t) { // 0-base
    vector<int> h((int)t.size());
    iota(all(h), 0);
    for (int a = 0; a < (int)s.size(); ++a) {
        int v = -1;
        for (int c = 0; c < (int)t.size(); ++c)
            if (s[a] == t[c] || h[c] < v)
                swap(h[c], v);
        // LCS(s[0, a], t[b, c]) =
        // c - b + 1 - sum([h[i] >= b] | i <= c)
        // h[i] might become -1 !!
    }
}
```

## 10.7 SimulatedAnnealing

```
double factor = 100000;
const int base = 1e9; // remember to run ~ 10 times
for (int it = 1; it <= 1000000; ++it) {
    // ans:
    answer, nw: current value, rnd(): mt19937 rnd()
    if (exp(-(nw - ans) / factor) >= (double)(rnd() % base) / base)
        ans = nw;
    factor *= 0.99995;
}
```

## 10.8 SMAWK

```
int opt[N];
ll A(int x, int y); // target func
void smawk(vector<int> &r, vector<int> &c);
void interpolate(vector<int> &r, vector<int> &c) {
    int n = (int)r.size();
    vector<int> er;
    for (int i = 1; i < n; i += 2) er.emplace_back(r[i]);
    smawk(er, c);
    for (int i = 0, j = 0; j < c.size(); j++) {
        if (A(r[i], c[j]) < A(r[i], opt[r[i]]))
            opt[r[i]] = c[j];
        if (i + 2 < n && c[j] == opt[r[i + 1]])
            j--, i += 2;
    }
}
void reduce(vector<int> &r, vector<int> &c) {
    int n = (int)r.size();
    vector<int> nc;
    for (int i : c) {
        int j = (int)nc.size();
        while (
            j && A(r[j - 1], nc[j - 1]) > A(r[j - 1], i))
            nc.pop_back(), j--;
        if (nc.size() < n) nc.emplace_back(i);
    }
    smawk(r, nc);
}
void smawk(vector<int> &r, vector<int> &c) {
    if (r.size() == 1 && c.size() == 1) opt[r[0]] = c[0];
    else if (r.size() >= c.size()) interpolate(r, c);
    else reduce(r, c);
}
```

## 10.9 Python

```
math.isqrt(2) # integer sqrt
```

## 10.10 LineContainer

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

```
struct LineContainer : multiset<Line, less<>> {
    // (for doubles, use inf = 1/.0, div(a,b) = a/b)
    static const ll inf = LLONG_MAX;
    ll div(ll a, ll b) { // floored division
        return a / b - ((a ^ b) < 0 && a % b);
    }
    bool isect(iterator x, iterator y) {
        if (y == end()) return x->p = inf, 0;
        if (x->k == y->k) x->p = x->m > y->m ? inf : -inf;
        else x->p = div(y->m - x->m, x->k - y->k);
        return x->p >= y->p;
    }
    void add(ll k, ll m) {
        auto z = insert({k, m, 0}), y = z++, x = y;
        while (isect(y, z)) z = erase(z);
        if (x != begin() && isect(--x, y))
            isect(x, y = erase(y));
        while ((y = x) != begin() && (--x)->p >= y->p)
            isect(x, erase(y));
    }
    ll query(ll x) {
        assert(!empty());
        auto l = *lower_bound(x);
        return l.k * x + l.m;
    }
};
```