

# Contents

<b>1 Basic</b>	<b>1</b>	6.12 QuadraticResidue*	13
1.1 vimrc	1	6.13 PiCount*	13
1.2 readchar	1	6.14 Discrete Log*	13
1.3 Black Magic	1	6.15 Berlekamp Massey	13
<b>2 Graph</b>	<b>1</b>	6.16 Primes	13
2.1 BCC Vertex*	1	6.17 Theorem	13
2.2 Bridge*	1	6.18 Estimation	14
2.3 2SAT (SCC)*	2	6.19 Euclidean Algorithms	14
2.4 Minimum Mean Cycle*	2	6.20 General Purpose Numbers	14
2.5 Virtual Tree*	2	6.21 Tips for Generating Functions	14
2.6 Maximum Clique Dyn*	2	<b>7 Polynomial</b>	<b>14</b>
2.7 Minimum Steiner Tree*	3	7.1 Fast Fourier Transform	14
2.8 Dominator Tree*	3	7.2 Number Theory Transform*	14
2.9 Minimum Arborescence*	3	7.3 Fast Walsh Transform*	15
2.10 Vizing's theorem*	4	7.4 Polynomial Operation	15
2.11 Minimum Clique Cover*	4	7.5 Value Polynomial	16
2.12 Number of Maximal Clique*	4	7.6 Newton's Method	16
<b>3 Data Structure</b>	<b>4</b>	<b>8 Geometry</b>	<b>16</b>
3.1 Discrete Trick	4	8.1 Default Code	16
3.2 Leftist Tree	4	8.2 PointSegDist*	16
3.3 Heavy light Decomposition	5	8.3 Heart	17
3.4 Centroid Decomposition*	5	8.4 point in circle	17
3.5 Link cut tree*	5	8.5 Convex hull*	17
3.6 KDTree	6	8.6 PointInConvex*	17
<b>4 Flow/Matching</b>	<b>6</b>	8.7 TangentPointToHull*	17
4.1 Kuhn Munkres*	6	8.8 Intersection of line and convex	17
4.2 MincostMaxflow*	7	8.9 minMaxEnclosingRectangle*	17
4.3 Maximum Simple Graph Matching*	7	8.10 VectorInPoly*	17
4.4 Minimum Weight Matching (Clique version)*	7	8.11 PolyUnion*	18
4.5 SW-mincut	8	8.12 Polar Angle Sort*	18
4.6 BoundedFlow*(Dinic*)	8	8.13 Half plane intersection*	18
4.7 Gomory Hu tree*	8	8.14 Rotating Sweep Line	18
4.8 Minimum Cost Circulation*	9	8.15 Minimum Enclosing Circle*	18
4.9 Flow Models	9	8.16 Intersection of two circles*	18
<b>5 String</b>	<b>9</b>	8.17 Intersection of polygon and circle*	19
5.1 KMP	9	8.18 Intersection of line and circle*	19
5.2 Z-value*	9	8.19 Tangent line of two circles	19
5.3 Manacher*	9	8.20 Circle Cover*	19
5.4 SAIS*	10	8.21 3D point*	19
5.5 Aho-Corasick Automaton	10	8.22 Convex hull 3D*	20
5.6 Smallest Rotation	10	8.23 Delaunay Triangulation*	20
5.7 De Bruijn sequence*	10	8.24 Triangulation Voronoi*	21
5.8 Extended SAM*	10	8.25 Minkowski Sum*	21
5.9 PalTree*	11	<b>9 Else</b>	<b>21</b>
<b>6 Math</b>	<b>11</b>	9.1 Cyclic Ternary Search*	21
6.1 ax+by=gcd(only exgcd*)	11	9.2 Mo's Algorithm (With modification)	21
6.2 Floor and Ceil	11	9.3 Mo's Algorithm On Tree	21
6.3 Floor Enumeration	11	9.4 Additional Mo's Algorithm Trick	22
6.4 Mod Min	11	9.5 Hilbert Curve	22
6.5 Gaussian integer gcd	11	9.6 Dynamic Convex Trick*	22
6.6 Miller Rabin*	11	9.7 All LCS*	22
6.7 Simultaneous Equations	12	9.8 DLX*	22
6.8 Pollard Rho*	12	9.9 Matroid Intersection	23
6.9 Simplex Algorithm	12	9.10 Adaptive Simpson	23
6.9.1 Construction	12	9.11 Simulated Annealing	23
6.10 chinese Remainder	12	9.12 Tree Hash*	23
6.11 Factorial without prime factor*	12	9.13 Binary Search On Fraction	23
		<b>10 Python</b>	<b>23</b>
		10.1 Misc	23

## 1 Basic

### 1.1 vimrc

"This file should be placed at ~/.vimrc"

```
se nu ai hls et ru ic is sc cul
se re=1 ts=4 sts=4 sw=4 ls=2 mouse=a
syntax on
hi cursorline cterm=none ctermbg=89
set bg=dark
inoremap {<CR> {<CR>}<Esc>ko<tab>
```

### 1.2 readchar

```
inline char readchar() {
    static const size_t bufsize = 65536;
    static char buf[bufsize];
    static char *p = buf, *end = buf;
    if (p == end) end = buf +
        fread_unlocked(buf, 1, bufsize, stdin), p = buf;
    return *p++;
}
```

### 1.3 Black Magic

```
#include <ext/pb_ds/priority_queue.hpp>
#include <ext/pb_ds/assoc_container.hpp> // rb_tree
```

```
#include <ext/rope> // rope
using namespace __gnu_pbds;
using namespace __gnu_cxx; // rope
typedef __gnu_pbds::priority_queue<int> heap;
int main() {
    heap h1, h2; // max heap
    h1.push(1), h1.push(3), h2.push(2), h2.push(4);
    h1.join(h2); // h1 = {1, 2, 3, 4}, h2 = {};
    tree<ll, null_type, less<ll>, rb_tree_tag,
        tree_order_statistics_node_update> st;
    tree<ll, ll, less<ll>, rb_tree_tag,
        tree_order_statistics_node_update> mp;
    for (int x : {0, 2, 3, 4}) st.insert(x);
    cout << *st.find_by_order
        (2) << st.order_of_key(1) << endl; //31
    rope<char> *root[10]; // nsqrt(n)
    root[0] = new rope<char>();
    root[1] = new rope<char>(*root[0]);
    // root[1]->insert(pos, 'a');
    // root[1]->at(pos); 0-base
    // root[1]->erase(pos, size);
}
// __int128_t, __float128_t
// for (int i = bs.Find_first
//     ()); i < bs.size(); i = bs._Find_next(i));
```

## 2 Graph

### 2.1 BCC Vertex\*

```
vector<int> G[N]; // 1-base
vector<int> nG[N * 2], bcc[N];
int low[N], dfn[N], Time;
int bcc_id[N], bcc_cnt; // 1-base
bool is_cut[N]; // whether is av
bool cir[N * 2];
int st[N], top;

void dfs(int u, int pa = -1) {
    int child = 0;
    low[u] = dfn[u] = ++Time;
    st[top++] = u;
    for (int v : G[u])
        if (!dfn[v]) {
            dfs(v, u), ++child;
            low[u] = min(low[u], low[v]);
            if (dfn[u] <= low[v]) {
                is_cut[u] = 1;
                bcc[++bcc_cnt].clear();
                int t;
                do {
                    bcc_id[t = st[--top]] = bcc_cnt;
                    bcc[bcc_cnt].push_back(t);
                } while (t != v);
                bcc_id[u] = bcc_cnt;
                bcc[bcc_cnt].pb(u);
            }
        } else if (dfn[v] < dfn[u] && v != pa)
            low[u] = min(low[u], dfn[v]);
    if (pa == -1 && child < 2) is_cut[u] = 0;
}

void bcc_init(int n) { // TODO: init {nG, cir}[1..2n]
    Time = bcc_cnt = top = 0;
    for (int i = 1; i <= n; ++i)
        G[i].clear(), dfn[i] = bcc_id[i] = is_cut[i] = 0;
}

void bcc_solve(int n) {
    for (int i = 1; i <= n; ++i)
        if (!dfn[i]) dfs(i);
    // block-cut tree
    for (int i = 1; i <= n; ++i)
        if (is_cut[i])
            bcc_id[i] = ++bcc_cnt, cir[bcc_cnt] = 1;
    for (int i = 1; i <= bcc_cnt && !cir[i]; ++i)
        for (int j : bcc[i])
            if (is_cut[j])
                nG[i].pb(bcc_id[j]), nG[bcc_id[j]].pb(i);
}
```

### 2.2 Bridge\*

```
int low[N], dfn[N], Time; // 1-base
vector<pii> G[N], edge;
vector<bool> is_bridge;
```

```

void init(int n) {
    Time = 0;
    for (int i = 1; i <= n; ++i)
        G[i].clear(), low[i] = dfn[i] = 0;
}

void add_edge(int a, int b) {
    G[a].pb(pii(b, SZ(edge))), G[b].pb(pii(a, SZ(edge)));
    edge.pb(pii(a, b));
}

void dfs(int u, int f) {
    dfn[u] = low[u] = ++Time;
    for (auto i : G[u])
        if (!dfn[i.X])
            dfs(i.X, i.Y), low[u] = min(low[u], low[i.X]);
        else if (i.Y != f) low[u] = min(low[u], dfn[i.X]);
    if (low[u] == dfn[u] && f != -1) is_bridge[f] = 1;
}

void solve(int n) {
    is_bridge.resize(SZ(edge));
    for (int i = 1; i <= n; ++i)
        if (!dfn[i]) dfs(i, -1);
}

```

## 2.3 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].pb(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[u], low[i]);
            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]].pb(i);
        for (int i = 0; i < n; ++i) {
            if (bln[i] == bln[i + n]) return false;
            istrue[i] = bln[i] < bln[i + n];
            istrue[i + n] = !istrue[i];
        }
        return true;
    }
};

```

## 2.4 MinimumMeanCycle\*

```

ll road[N][N]; // input here
struct MinimumMeanCycle {
    ll dp[N + 5][N], n;
    pll solve() {
        ll a = -1, b = -1, L = n + 1;

```

```

        for (int i = 2; i <= L; ++i)
            for (int k = 0; k < n; ++k)
                for (int j = 0; j < n; ++j)
                    dp[i][j] =
                        min(dp[i - 1][k] + road[k][j], dp[i][j]);
        for (int i = 0; i < n; ++i) {
            if (dp[L][i] >= INF) continue;
            ll ta = 0, tb = 1;
            for (int j = 1; j < n; ++j)
                if (dp[j][i] < INF &&
                    ta * (L - j) < (dp[L][i] - dp[j][i]) * tb)
                    ta = dp[L][i] - dp[j][i], tb = L - j;
            if (ta == 0) continue;
            if (a == -1 || a * tb > ta * b) a = ta, b = tb;
        }
        if (a != -1) {
            ll g = __gcd(a, b);
            return pll(a / g, b / g);
        }
        return pll(-1LL, -1LL);
    }
    void init(int _n) {
        n = _n;
        for (int i = 0; i < n; ++i)
            for (int j = 0; j < n; ++j) dp[i + 2][j] = INF;
    }
};

```

## 2.5 Virtual Tree\*

```

vector<int> vG[N];
int top, st[N];

void insert(int u) {
    if (top == -1) return st[++top] = u, void();
    int p = LCA(st[top], u);
    if (p == st[top]) return st[++top] = u, void();
    while (top >= 1 && dep[st[top - 1]] >= dep[p])
        vG[st[top - 1]].pb(st[top]), --top;
    if (st[top] != p)
        vG[p].pb(st[top]), --top, st[++top] = p;
    st[++top] = u;
}

void reset(int u) {
    for (int i : vG[u]) reset(i);
    vG[u].clear();
}

void solve(vector<int> &v) {
    top = -1;
    sort(ALL(v),
        [&](int a, int b) { return dfn[a] < dfn[b]; });
    for (int i : v) insert(i);
    while (top > 0) vG[st[top - 1]].pb(st[top]), --top;
    // do something
    reset(v[0]);
}

```

## 2.6 Maximum Clique Dyn\*

```

struct MaxClique { // fast when N <= 100
    bitset<N> G[N], cs[N];
    int ans, sol[N], q, cur[N], d[N], n;
    void init(int _n) {
        n = _n;
        for (int i = 0; i < n; ++i) G[i].reset();
    }
    void add_edge(int u, int v) {
        G[u][v] = G[v][u] = 1;
    }
    void pre_dfs(vector<int> &r, int l, bitset<N> mask) {
        if (l < 4) {
            for (int i : r) d[i] = (G[i] & mask).count();
            sort(ALL(r),
                [&](int x, int y) { return d[x] > d[y]; });
        }
        vector<int> c(SZ(r));
        int lft = max(ans - q + 1, 1), rgt = 1, tp = 0;
        cs[1].reset(), cs[2].reset();
        for (int p : r) {
            int k = 1;
            while ((cs[k] & G[p]).any()) ++k;
            if (k > rgt) cs[++rgt + 1].reset();
            cs[k][p] = 1;
            if (k < lft) r[tp++] = p;
        }
    }
};

```

```

    }
    for (int k = lft; k <= rgt; ++k)
        for (int p = cs[k]._Find_first
              (); p < N; p = cs[k]._Find_next(p))
            r[tp] = p, c[tp] = k, ++tp;
    dfs(r, c, l + 1, mask);
}
void dfs(vector<
    int> &r, vector<int> &c, int l, bitset<N> mask) {
    while (!r.empty()) {
        int p = r.back();
        r.pop_back(), mask[p] = 0;
        if (q + c.back() <= ans) return;
        cur[q++] = p;
        vector<int> nr;
        for (int i : r) if (G[p][i]) nr.pb(i);
        if (!nr.empty()) pre_dfs(nr, l, mask & G[p]);
        else if (q > ans) ans = q, copy_n(cur, q, sol);
        c.pop_back(), --q;
    }
}
int solve() {
    vector<int> r(n);
    ans = q = 0, iota(ALL(r), 0);
    pre_dfs(r, 0, bitset<N>(string(n, '1')));
    return ans;
}
};

```

## 2.7 Minimum Steiner Tree\*

```

struct SteinerTree { // 0-base
    int n, dst[N][N], dp[1 << T][N], tdst[N];
    int vcst[N]; // the cost of vertexs
    void init(int _n) {
        n = _n;
        for (int i = 0; i < n; ++i) {
            fill_n(dst[i], n, INF);
            dst[i][i] = vcst[i] = 0;
        }
    }
    void chmin(int &x, int val) {
        x = min(x, val);
    }
    void add_edge(int ui, int vi, int wi) {
        chmin(dst[ui][vi], wi);
    }
    void shortest_path() {
        for (int k = 0; k < n; ++k)
            for (int i = 0; i < n; ++i)
                for (int j = 0; j < n; ++j)
                    chmin(dst[i][j], dst[i][k] + dst[k][j]);
    }
    int solve(const vector<int> &ter) {
        shortest_path();
        int t = SZ(ter), full = (1 << t) - 1;
        for (int i = 0; i <= full; ++i)
            fill_n(dp[i], n, INF);
        copy_n(vkst, n, dp[0]);
        for (int msk = 1; msk <= full; ++msk) {
            if (!(msk & (msk - 1))) {
                int who = __lg(msk);
                for (int i = 0; i < n; ++i)
                    dp[msk][i] = vcst[ter[who]] + dst[ter[who]][i];
            }
            for (int i = 0; i < n; ++i)
                for (int sub = (
                    msk - 1) & msk; sub; sub = (sub - 1) & msk)
                    chmin(dp[msk][i],
                        dp[sub][i] + dp[msk ^ sub][i] - vcst[i]);
            for (int i = 0; i < n; ++i) {
                tdst[i] = INF;
                for (int j = 0; j < n; ++j)
                    chmin(tdst[i], dp[msk][j] + dst[j][i]);
            }
            copy_n(tdst, n, dp[msk]);
        }
        return *min_element(dp[full], dp[full] + n);
    }
}; // O(V 3^T + V^2 2^T)

```

## 2.8 Dominator Tree\*

```

struct dominator_tree { // 1-base
    vector<int> G[N], rG[N];

```

```

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

```

## 2.9 Minimum Arborescence\*

```

/* TODO
DSU: disjoint set
- DSU(n), .boss(x), .Union(x, y)
min_heap<
    T, Info>: min heap for type {T, Info} with lazy tag
- .push({w, i}),
  .top(), .join(heap), .pop(), .empty(), .add_lazy(v)
*/
struct E { int s, t; ll w; }; // 0-base
vector<int> dmst(const vector<E> &e, int n, int root) {
    vector<min_heap<ll, int>> h(n * 2);
    for (int i = 0; i < SZ(e); ++i)
        h[e[i].t].push({e[i].w, i});
    DSU dsu(n * 2);
    vector<int> v(n * 2, -1), pa(n * 2, -1), r(n * 2);
    v[root] = n + 1;
    int pc = n;
    for (int i = 0; i < n; ++i) if (v[i] == -1) {
        for (int p = i; v[p]
            == -1 || v[p] == i; p = dsu.boss(e[r[p]].s)) {
            if (v[p] == i) {
                int q = p; p = pc++;
                do {
                    h[q].add_lazy(-h[q].top().X);
                    pa[q] = p, dsu.Union(p, q), h[p].join(h[q]);
                } while ((q = dsu.boss(e[r[q]].s)) != p);
            }
            v[p] = i;
            while (!h[p].
                empty()) && dsu.boss(e[h[p].top().Y].s) == p)
                h[p].pop();

```

```

    if (h[p].empty()) return {}; // no solution
    r[p] = h[p].top().Y;
}
}
vector<int> ans;
for (int i = pc
    - 1; i >= 0; i--) if (i != root && v[i] != n) {
    for (int f = e[r[i]].t; ~f && v[f] != n; f = pa[f])
        v[f] = n;
    ans.pb(r[i]);
}
return ans; // default minimize, returns edgeid array
} // O(Ef(E)), f(E) from min_heap

```

## 2.10 Vizing's theorem\*

```

namespace vizing { // returns
    edge coloring in adjacent matrix G. 1 - based
const int N = 105;
int C[N][N], G[N][N], X[N], vst[N], n;
void init(int _n) { n = _n;
    for (int i = 0; i <= n; ++i)
        for (int j = 0; j <= n; ++j)
            C[i][j] = G[i][j] = 0;
}
void solve(vector<pii> &E) {
    auto update = [&](int u)
    { for (X[u] = 1; C[u][X[u]]; ++X[u]); };
    auto color = [&](int u, int v, int c) {
        int p = G[u][v];
        G[u][v] = G[v][u] = c;
        C[u][c] = v, C[v][c] = u;
        C[u][p] = C[v][p] = 0;
        if (p) X[u] = X[v] = p;
        else update(u), update(v);
        return p;
    };
    auto flip = [&](int u, int c1, int c2) {
        int p = C[u][c1];
        swap(C[u][c1], C[u][c2]);
        if (p) G[u][p] = G[p][u] = c2;
        if (!C[u][c1]) X[u] = c1;
        if (!C[u][c2]) X[u] = c2;
        return p;
    };
    fill_n(X + 1, n, 1);
    for (int t = 0; t < SZ(E); ++t) {
        int u = E[t].X, v0 = E[t].Y, v = v0, c0 = X[u], c = c0, d;
        vector<pii> L;
        fill_n(vst + 1, n, 0);
        while (!G[u][v0]) {
            L.emplace_back(v, d = X[v]);
            if (!C[v][c]) for (int a = SZ(L) - 1; a >= 0; --a) c = color(u, L[a].X, c);
            else if (!C[u][d]) for (int a = SZ(L) - 1; a >= 0; --a) color(u, L[a].X, L[a].Y);
            else if (vst[d]) break;
            else vst[d] = 1, v = C[u][d];
        }
        if (!G[u][v0]) {
            for (; v; v = flip(v, c, d), swap(c, d));
            if (int a; C[u][c0]) {
                for (
                    a = SZ(L) - 2; a >= 0 && L[a].Y != c; --a);
                for (; a >= 0; --a) color(u, L[a].X, L[a].Y);
            }
            else --t;
        }
    }
}
} // namespace vizing

```

## 2.11 Minimum Clique Cover\*

```

struct Clique_Cover { // 0-base, O(n2^n)
    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;
    }
};

```

## 2.12 NumberofMaximalClique\*

```

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 tsn = 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][tsn++] = 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, tsn, 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;
    }
};

```

## 3 Data Structure

### 3.1 Discrete Trick

```

vector<int> val;
// build
sort(ALL
    (val)), val.resize(unique(ALL(val)) - val.begin());
// index of x
upper_bound(ALL(val), x) - val.begin();
// max idx <= x
upper_bound(ALL(val), x) - val.begin();
// max idx < x
lower_bound(ALL(val), x) - val.begin();

```

### 3.2 Leftist Tree

```

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

```

```

ll V(node *p) { return p ? p->v : -1; }
ll sum(node *p) { return p ? p->sum : 0; }
node *merge(node *a, node *b) {
    if (!a || !b) return a ? a : b;
    if (a->data < b->data) swap(a, b);
    a->r = merge(a->r, b);
    if (V(a->r) > V(a->l)) swap(a->r, a->l);
    a->v = V(a->r) + 1, a->sz = sz(a->l) + sz(a->r) + 1;
    a->sum = sum(a->l) + sum(a->r) + a->data;
    return a;
}
void pop(node *&o) {
    node *tmp = o;
    o = merge(o->l, o->r);
    delete tmp;
}

```

### 3.3 Heavy light Decomposition

```

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;
    }
};

```

### 3.4 Centroid Decomposition\*

```

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;
        }
    };
};

```

### 3.5 Link cut tree\*

```

struct Splay { // xor-sum
    static Splay nil;
    Splay *ch[2], *f;
    int val, sum, rev, size;
    Splay(int _val = 0)
        : val(_val), sum(_val), rev(0), size(1) {
        f = ch[0] = ch[1] = &nil;
    }
    bool isr() {
        return f->ch[0] != this && f->ch[1] != this;
    }
    int dir() { return f->ch[0] == this ? 0 : 1; }
    void setCh(Splay *c, int d) {
        ch[d] = c;
        if (c != &nil) c->f = this;
        pull();
    }
    void give_tag(int r)
        { if (r) swap(ch[0], ch[1]), rev ^= 1; }
    void push() {
        if (ch[0] != &nil) ch[0]->give_tag(rev);
        if (ch[1] != &nil) ch[1]->give_tag(rev);
        rev = 0;
    }
    void pull() {
        // take care of the nil!
        size = ch[0]->size + ch[1]->size + 1;
        sum = ch[0]->sum ^ ch[1]->sum ^ val;
        if (ch[0] != &nil) ch[0]->f = this;
        if (ch[1] != &nil) ch[1]->f = this;
    }
} Splay::nil;
Splay *nil = &Splay::nil;
void rotate(Splay *x) {
    Splay *p = x->f;
    int d = x->dir();
    if (!p->isr()) p->f->setCh(x, p->dir());

```

```

else x->f = p->f;
p->setCh(x->ch[!d], d);
x->setCh(p, !d);
p->pull(), x->pull();
}
void splay(Splay *x) {
    vector<Splay *> splayVec;
    for (Splay *q = x;; q = q->f) {
        splayVec.pb(q);
        if (q->isr()) break;
    }
    reverse(ALL(splayVec));
    for (auto it : splayVec) it->push();
    while (!x->isr()) {
        if (x->f->isr()) rotate(x);
        else if (x->dir() == x->f->dir())
            rotate(x->f), rotate(x);
        else rotate(x), rotate(x);
    }
}
Splay *access(Splay *x) {
    Splay *q = nil;
    for (; x != nil; x = x->f)
        splay(x), x->setCh(q, 1), q = x;
    return q;
}
void root_path(Splay *x) { access(x), splay(x); }
void chroot(Splay *x) {
    root_path(x), x->rev ^= 1;
    x->push(), x->pull();
}
void split(Splay *x, Splay *y) {
    chroot(x), root_path(y);
}
void link(Splay *x, Splay *y) {
    root_path(x), chroot(y);
    x->setCh(y, 1);
}
void cut(Splay *x, Splay *y) {
    split(x, y);
    if (y->size != 5) return;
    y->push();
    y->ch[0] = y->ch[0]->f = nil;
}
Splay *get_root(Splay *x) {
    for (root_path(x); x->ch[0] != nil; x = x->ch[0])
        x->push();
    splay(x);
    return x;
}
bool conn(Splay *x, Splay *y) {
    return get_root(x) == get_root(y);
}
Splay *lca(Splay *x, Splay *y) {
    access(x), root_path(y);
    if (y->f == nil) return y;
    return y->f;
}
void change(Splay *x, int val) {
    splay(x), x->val = val, x->pull();
}
int query(Splay *x, Splay *y) {
    split(x, y);
    return y->sum;
}
}

```

### 3.6 KDTree

```

namespace kdt {
int root, lc[maxn], rc[maxn], xl[maxn], xr[maxn],
    yl[maxn], yr[maxn];
point p[maxn];
int build(int l, int r, int dep = 0) {
    if (l == r) return -1;
    function<bool(const point &, const point &)> f =
        [dep](const point &a, const point &b) {
            if (dep & 1) return a.x < b.x;
            else return a.y < b.y;
        };
    int m = (l + r) >> 1;
    nth_element(p + l, p + m, p + r, f);
    xl[m] = xr[m] = p[m].x;
    yl[m] = yr[m] = p[m].y;
    lc[m] = build(l, m, dep + 1);
    if (~lc[m]) {
        xl[m] = min(xl[m], xl[lc[m]]);

```

```

        xr[m] = max(xr[m], xr[lc[m]]);
        yl[m] = min(yl[m], yl[lc[m]]);
        yr[m] = max(yr[m], yr[lc[m]]);
    }
    rc[m] = build(m + 1, r, dep + 1);
    if (~rc[m]) {
        xl[m] = min(xl[m], xl[rc[m]]);
        xr[m] = max(xr[m], xr[rc[m]]);
        yl[m] = min(yl[m], yl[rc[m]]);
        yr[m] = max(yr[m], yr[rc[m]]);
    }
    return m;
}
bool bound(const point &q, int o, long long d) {
    double ds = sqrt(d + 1.0);
    if (q.x < xl[o] - ds || q.x > xr[o] + ds ||
        q.y < yl[o] - ds || q.y > yr[o] + ds)
        return false;
    return true;
}
long long dist(const point &a, const point &b) {
    return (a.x - b.x) * 1ll * (a.x - b.x) +
        (a.y - b.y) * 1ll * (a.y - b.y);
}
void dfs(
    const point &q, long long &d, int o, int dep = 0) {
    if (!bound(q, o, d)) return;
    long long cd = dist(p[o], q);
    if (cd != 0) d = min(d, cd);
    if ((dep & 1) && q.x < p[o].x ||
        !(dep & 1) && q.y < p[o].y) {
        if (~lc[o]) dfs(q, d, lc[o], dep + 1);
        if (~rc[o]) dfs(q, d, rc[o], dep + 1);
    } else {
        if (~rc[o]) dfs(q, d, rc[o], dep + 1);
        if (~lc[o]) dfs(q, d, lc[o], dep + 1);
    }
}
void init(const vector<point> &v) {
    for (int i = 0; i < v.size(); ++i) p[i] = v[i];
    root = build(0, v.size());
}
long long nearest(const point &q) {
    long long res = 1e18;
    dfs(q, res, root);
    return res;
}
} // namespace kdt

```

## 4 Flow/Matching

### 4.1 Kuhn Munkres\*

```

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;
}
};

```

## 4.2 MincostMaxflow\*

```

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].pb(Edge{a, b, cap, 0, cost, SZ(G[b])});
        G[b].pb(Edge{b, a, 0, 0, -cost, SZ(G[a]) - 1});
    }
};

```

## 4.3 Maximum Simple Graph Matching\*

```

struct GenMatch { // 1-base
    int V, pr[N];
    bool el[N][N], inq[N], inp[N], inb[N];
    int st, ed, nb, bk[N], djs[N], ans;
    void init(int _V) {
        V = _V;
        for (int i = 0; i <= V; ++i) {
            for (int j = 0; j <= V; ++j) el[i][j] = 0;
            pr[i] = bk[i] = djs[i] = 0;
        }
    }
};

```

```

    inq[i] = inp[i] = inb[i] = 0;
}
}
void add_edge(int u, int v) {
    el[u][v] = el[v][u] = 1;
}
int lca(int u, int v) {
    fill_n(inp, V + 1, 0);
    while (1)
        if (u = djs[u], inp[u] = true, u == st) break;
        else u = bk[pr[u]];
    while (1)
        if (v = djs[v], inp[v]) return v;
        else v = bk[pr[v]];
    return v;
}
void upd(int u) {
    for (int v; djs[u] != nb;) {
        v = pr[u], inb[djs[u]] = inb[djs[v]] = true;
        u = bk[v];
        if (djs[u] != nb) bk[u] = v;
    }
}
void blo(int u, int v, queue<int> &qe) {
    nb = lca(u, v), fill_n(inb, V + 1, 0);
    upd(u), upd(v);
    if (djs[u] != nb) bk[u] = v;
    if (djs[v] != nb) bk[v] = u;
    for (int tu = 1; tu <= V; ++tu)
        if (inb[djs[tu]])
            if (djs[tu] == nb, !inq[tu])
                qe.push(tu), inq[tu] = 1;
}
void flow() {
    fill_n(inq + 1, V, 0), fill_n(bk + 1, V, 0);
    iota(djs + 1, djs + V + 1, 1);
    queue<int> qe;
    qe.push(st), inq[st] = 1, ed = 0;
    while (!qe.empty()) {
        int u = qe.front();
        qe.pop();
        for (int v = 1; v <= V; ++v)
            if (el[u][v] && djs[u] != djs[v] &&
                pr[u] != v) {
                if ((v == st) ||
                    (pr[v] > 0 && bk[pr[v]] > 0)) {
                    blo(u, v, qe);
                } else if (!bk[v]) {
                    if (bk[v] = u, pr[v] > 0) {
                        if (!inq[pr[v]]) qe.push(pr[v]);
                    } else {
                        return ed = v, void();
                    }
                }
            }
    }
}
void aug() {
    for (int u = ed, v, w; u > 0;)
        v = bk[u], w = pr[v], pr[v] = u, pr[u] = v,
        u = w;
}
int solve() {
    fill_n(pr, V + 1, 0), ans = 0;
    for (int u = 1; u <= V; ++u)
        if (!pr[u])
            if (st = u, flow(), ed > 0) aug(), ++ans;
    return ans;
}
};

```

## 4.4 Minimum Weight Matching (Clique version)\*

```

struct Graph { // 0-base (Perfect Match), n is even
    int n, match[N], onstk[N], stk[N], tp;
    ll edge[N][N], dis[N];
    void init(int _n) {
        n = _n, tp = 0;
        for (int i = 0; i < n; ++i) fill_n(edge[i], n, 0);
    }
    void add_edge(int u, int v, ll w) {
        edge[u][v] = edge[v][u] = w;
    }
    bool SPFA(int u) {
        stk[tp++] = u, onstk[u] = 1;
    }
};

```

```

for (int v = 0; v < n; ++v)
    if (!onstk[v] && match[u] != v) {
        int m = match[v];
        if (dis[m] >
            dis[u] - edge[v][m] + edge[u][v]) {
            dis[m] = dis[u] - edge[v][m] + edge[u][v];
            onstk[v] = 1, stk[tp++] = v;
            if (onstk[m] || SPFA(m)) return 1;
            --tp, onstk[v] = 0;
        }
    }
onstk[u] = 0, --tp;
return 0;
}
ll solve() { // find a match
for (int i = 0; i < n; ++i) match[i] = i ^ 1;
while (1) {
    int found = 0;
    fill_n(dis, n, 0);
    fill_n(onstk, n, 0);
    for (int i = 0; i < n; ++i)
        if (tp = 0, !onstk[i] && SPFA(i))
            for (found = 1; tp >= 2; ) {
                int u = stk[--tp];
                int v = stk[--tp];
                match[u] = v, match[v] = u;
            }
    if (!found) break;
}
ll ret = 0;
for (int i = 0; i < n; ++i)
    ret += edge[i][match[i]];
return ret >> 1;
}
};

```

## 4.5 SW-mincut

```

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;

```

## 4.6 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].pb(edge{v, rcap, lcap, SZ(G[v])});
        G[v].pb(edge{u, 0, 0, SZ(G[u]) - 1});
    }
    void add_edge(int u, int v, int cap) {
        G[u].pb(edge{v, cap, 0, SZ(G[v])});
        G[v].pb(edge{u, 0, 0, SZ(G[u]) - 1});
    }
    int dfs(int u, int cap) {
        if (u == t || !cap) return cap;
        for (int &i = cur[u]; i < SZ(G[u]); ++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;
    }
};

```

## 4.7 Gomory Hu tree\*

```

MaxFlow Dinic;
int g[MXN];
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;
    }
}

```



## 4.8 Minimum Cost Circulation\*

```

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].pb(Edge{a, b, 0, cap, 0, cost, SZ(G[b]) + (a == b)});
            G[b].pb(Edge{b, a, 0, 0, 0, -cost, SZ(G[a]) - 1});
        }
    } mcmf; // O(VE * ElogC)
}

```

## 4.9 Flow Models

- 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.

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

## 5 String

### 5.1 KMP

```

int F[MAXN];
vector<int> match(string A, string B) {
    vector<int> ans;
    F[0] = -1, F[1] = 0;
    for (int i = 1, j = 0; i < SZ(B); F[++i] = ++j) {
        if (B[i] == B[j]) F[i] = F[j]; // optimize
        while (j != -1 && B[i] != B[j]) j = F[j];
    }
    for (int i = 0, j = 0; i < SZ(A); ++i) {
        while (j != -1 && A[i] != B[j]) j = F[j];
        if (++j == SZ(B)) ans.pb(i + 1 - j), j = F[j];
    }
    return ans;
}

```

### 5.2 Z-value\*

```

int z[MAXN];
void make_z(const string &s) {
    int l = 0, r = 0;
    for (int i = 1; i < SZ(s); ++i) {
        for (z[i] = max(0, min(r - i + 1, z[i - l]));
             i + z[i] < SZ(s) && s[i + z[i]] == s[z[i]];
             ++z[i])
            ;
        if (i + z[i] - 1 > r) l = i, r = i + z[i] - 1;
    }
}

```

### 5.3 Manacher\*

```

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

```

## 5.4 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 *sa, 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);
    }
}
```

## 5.5 Aho-Corasick Automatan

```
const int len = 400000, sigma = 26;
struct AC_Automatan {
    int nx[len][sigma], fl[len], cnt[len], pri[len], top;
    int newnode() {
        fill(nx[top], nx[top] + sigma, -1);
        return top++;
    }
    void init() { top = 1, newnode(); }
    int input(
        string &s) { // return the end_node of string
        int X = 1;
        for (char c: s) {
```

```
            if (!~nx[X][c - 'a']) nx[X][c - 'a'] = newnode();
            X = nx[X][c - 'a'];
        }
        return X;
    }
    void make_fl() {
        queue<int> q;
        q.push(1), fl[1] = 0;
        for (int t = 0; !q.empty(); ) {
            int R = q.front();
            q.pop(), pri[t++] = R;
            for (int i = 0; i < sigma; ++i)
                if (~nx[R][i]) {
                    int X = nx[R][i], Z = fl[R];
                    for (; Z && !~nx[Z][i];) Z = fl[Z];
                    fl[X] = Z ? nx[Z][i] : 1, q.push(X);
                }
        }
    }
    void get_v(string &s) {
        int X = 1;
        fill(cnt, cnt + top, 0);
        for (char c: s) {
            while (X && !~nx[X][c - 'a']) X = fl[X];
            X = X ? nx[X][c - 'a'] : 1, ++cnt[X];
        }
        for (int i = top - 2; i > 0; --i)
            cnt[fl[pri[i]]] += cnt[pri[i]];
    }
};
```

## 5.6 Smallest Rotation

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

## 5.7 De Bruijn sequence\*

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

## 5.8 Extended SAM\*

```
struct exSAM {
    int len[N * 2], link[N * 2]; // maxlength, suflink
    int next[N * 2][CNUM], tot; // [0, tot), root = 0
    int lenSorted[N * 2]; // topo. order
    int cnt[N * 2]; // occurence
    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;
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 PalTree\*

```

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.pb(-1);
    }
    inline void clear() {
        St.clear(), s.clear(), last = 1, n = 0;
        St.pb(0), St.pb(-1);
        St[0].fail = 1, s.pb(-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 = SZ(St);
            St.pb(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 SZ(St) - 2;
}
};

```

## 6 Math

### 6.1 ax+by=gcd(only exgcd\*)

```

pll exgcd(ll a, ll b) {
    if (b == 0) return pll(1, 0);
    ll p = a / b;
    pll q = exgcd(b, a % b);
    return pll(q.Y, q.X - q.Y * p);
}
/* ax+by=res, let x be minimum non-negative
g, p = gcd(a, b), exgcd(a, b) * res / g
if p.X < 0: t = (abs(p.X) + b / g - 1) / (b / g)
else: t = -(p.X / (b / g))
p += (b / g, -a / g) * t */

```

### 6.2 Floor and Ceil

```

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)); }

```

### 6.3 Floor Enumeration

```

// enumerating x = floor(n / i), [l, r]
for (int l = 1, r; l <= n; l = r + 1) {
    int x = n / l;
    r = n / x;
}

```

### 6.4 Mod Min

```

// min{k | l <= ((ak) mod m) <= r}, no solution -> -1
ll mod_min(ll a, ll m, ll l, ll r) {
    if (a == 0) return l ? -1 : 0;
    if (ll k = (l + a - 1) / a; k * a <= r)
        return k;
    ll b = m / a, c = m % a;
    if (ll y = mod_min(c, a, a - r % a, a - l % a))
        return (l + y * c + a - 1) / a + y * b;
    return -1;
}

```

### 6.5 Gaussian integer gcd

```

cpx gaussian_gcd(cpx a, cpx b) {
#define rnd
    (a, b) ((a >= 0 ? a * 2 + b : a * 2 - b) / (b * 2))
    ll c = a.real() * b.real() + a.imag() * b.imag();
    ll d = a.imag() * b.real() - a.real() * b.imag();
    ll r = b.real() * b.real() + b.imag() * b.imag();
    if (c % r == 0 && d % r == 0) return b;
    return gaussian_gcd
        (b, a - cpx(rnd(c, r), rnd(d, r)) * b);
}

```

### 6.6 Miller Rabin\*

```

// 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 : pimes <= 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;
}

```

## 6.7 Simultaneous Equations

```

struct matrix { //m variables, n equations
    int n, m;
    fraction M[MAXN][MAXN + 1], sol[MAXN];
    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;
    }
};

```

## 6.8 Pollard Rho\*

```

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);
    }
}

```

## 6.9 Simplex Algorithm

```

const int MAXN = 11000, MAXM = 405;
const double eps = 1E-10;
double a[MAXN][MAXM], b[MAXN], c[MAXM];
double d[MAXN][MAXM], x[MAXM];
int ix[MAXN + MAXM]; // !!! array all indexed from 0
// max{cx} subject to {Ax<=b, x>=0}
// n: constraints, m: vars !!!
// x[] is the optimal solution vector
// usage :
// value = simplex(a, b, c, N, M);
double simplex(int n, int m){
    ++m;
    fill_n(d[n], m + 1, 0);
    fill_n(d[n + 1], m + 1, 0);
    iota(ix, ix + n + m, 0);
    int r = n, s = m - 1;
    for (int i = 0; i < n; ++i) {
        for (int j = 0; j < m - 1; ++j) d[i][j] = -a[i][j];
        d[i][m - 1] = 1;
        d[i][m] = b[i];
        if (d[r][m] > d[i][m]) r = i;
    }
    copy_n(c, m - 1, d[n]);
    d[n + 1][m - 1] = -1;
    for (double dd; ) {
        if (r < n) {
            swap(ix[s], ix[r + m]);

```

```

d[r][s] = 1.0 / d[r][s];
for (int j = 0; j <= m; ++j)
    if (j != s) d[r][j] *= -d[r][s];
for (int i = 0; i <= n + 1; ++i) if (i != r) {
    for (int j = 0; j <= m; ++j) if (j != s)
        d[i][j] += d[r][j] * d[i][s];
    d[i][s] *= d[r][s];
}
}
r = s = -1;
for (int j = 0; j < m; ++j)
    if (s < 0 || ix[s] > ix[j]) {
        if (d[n + 1][j] > eps ||
            (d[n + 1][j] > -eps && d[n][j] > eps))
            s = j;
    }
if (s < 0) break;
for (int i = 0; i < n; ++i) if (d[i][s] < -eps) {
    if (r < 0 ||
        (dd = d[r][m]
            / d[r][s] - d[i][m] / d[i][s]) < -eps ||
        (dd < eps && ix[r + m] > ix[i + m]))
        r = i;
}
if (r < 0) return -1; // not bounded
}
if (d[n + 1][m] < -eps) return -1; // not executable
double ans = 0;
fill_n(x, m, 0);
for (int i = m; i <
    n + m; ++i) { // the missing enumerated x[i] = 0
    if (ix[i] < m - 1) {
        ans += d[i - m][m] * c[ix[i]];
        x[ix[i]] = d[i - m][m];
    }
}
return ans;
}

```

### 6.9.1 Construction

Standard form: maximize  $\mathbf{c}^T \mathbf{x}$  subject to  $A\mathbf{x} \leq \mathbf{b}$  and  $\mathbf{x} \geq 0$ .

Dual LP: minimize  $\mathbf{b}^T \mathbf{y}$  subject to  $A^T \mathbf{y} \geq \mathbf{c}$  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  $x_i - x'_i$

## 6.10 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;
    pll p = exgcd(m1, m2);
    ll lcm = m1 * m2 * g;
    ll res = p.first * (x2 - x1) * m1 + x1;
    // be careful with overflow
    return (res % lcm + lcm) % lcm;
}

```

## 6.11 Factorial without prime factor\*

```

// 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

```

## 6.12 QuadraticResidue\*

```
int Jacobi(int a, int m) {
    int s = 1;
    for (; m > 1; ) {
        a %= m;
        if (a == 0) return 0;
        const int r = __builtin_ctz(a);
        if ((r & 1) && ((m + 2) & 4)) s = -s;
        a >>= r;
        if (a & m & 2) s = -s;
        swap(a, m);
    }
    return s;
}

int QuadraticResidue(int a, int p) {
    if (p == 2) return a & 1;
    const int jc = Jacobi(a, p);
    if (jc == 0) return 0;
    if (jc == -1) return -1;
    int b, d;
    for (; ; ) {
        b = rand() % p;
        d = (1LL * b * b + p - a) % p;
        if (Jacobi(d, p) == -1) break;
    }
    int f0 = b, f1 = 1, g0 = 1, g1 = 0, tmp;
    for (int e = (1LL + p) >> 1; e; e >>= 1) {
        if (e & 1) {
            tmp = (1LL *
                g0 * f0 + 1LL * d * (1LL * g1 * f1 % p)) % p;
            g1 = (1LL * g0 * f1 + 1LL * g1 * f0) % p;
            g0 = tmp;
        }
        tmp = (1LL
            * f0 * f0 + 1LL * d * (1LL * f1 * f1 % p)) % p;
        f1 = (2LL * f0 * f1) % p;
        f0 = tmp;
    }
    return g0;
}
```

## 6.13 PiCount\*

```
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];
}
```

## 6.14 Discrete Log\*

```
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;
}
```

## 6.15 Berlekamp Massey

```
template <typename T>
vector<T> BerlekampMassey(const vector<T> &output) {
    vector<T> d(SZ(output) + 1), me, he;
    for (int f = 0, i = 1; i <= SZ(output); ++i) {
        for (int j = 0; j < SZ(me); ++j)
            d[i] += output[i - j - 2] * 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.pb(-k);
        for (T x : he) o.pb(x * k);
        o.resize(max(SZ(o), SZ(me)));
        for (int j = 0; j < SZ(me); ++j) o[j] += me[j];
        if (i - f + SZ(he) >= SZ(me)) he = me, f = i;
        me = o;
    }
    return me;
}
```

## 6.16 Primes

```
/* 12721 13331 14341 75577 123457 222557
   556679 999983 1097774749 1076767633 100102021
   999997771 1001010013 1000512343 987654361 999991231
   999888733 98789101 987777733 999991921 1010101333
   1010102101 1000000000039 1000000000000037
   2305843009213693951 4611686018427387847
   9223372036854775783 18446744073709551557 */
```

## 6.17 Theorem

- Cramer's rule

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

- 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  $\sum_{i=1}^k d_i \leq k(k-1) + \sum_{i=k+1}^n \min(d_i, k)$  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  $\sum_{i=1}^k a_i \leq \sum_{i=1}^k \min(b_i, k)$  holds for every  $1 \leq k \leq n$ .
- Fulkerson–Chen–Anstee theorem  
A sequence  $(a_1, b_1), \dots, (a_n, b_n)$  of nonnegative integer pairs with  $a_1 \geq \dots \geq a_n$  is digraphic if and only if  $\sum_{i=1}^n a_i = \sum_{i=1}^n b_i$  and  $\sum_{i=1}^k a_i \leq \sum_{i=1}^k \min(b_i, k-1) + \sum_{i=k+1}^n \min(b_i, k)$  holds for every  $1 \leq k \leq n$ .
- Möbius inversion formula
  - $f(n) = \sum_{d|n} g(d) \Leftrightarrow g(n) = \sum_{d|n} \mu(d) f(\frac{n}{d})$
  - $f(n) = \sum_{n|d} g(d) \Leftrightarrow g(n) = \sum_{n|d} \mu(\frac{d}{n}) f(d)$
- 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  $= 2\pi r h = \pi(a^2 + h^2) = 2\pi r^2 (1 - \cos\theta)$ .
- 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.

## 6.18 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:  $B(n) = 1, 1, 2, 5, 15, 52, 203, 877, 4140, 21147, 115975, 678570, 4213597, 27644437, 190899322, \dots$

## 6.19 Euclidean Algorithms

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

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

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

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

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

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

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

## 6.20 General Purpose Numbers

- Bernoulli numbers  
 $B_0 = 1, B_1^\pm = \pm \frac{1}{2}, B_2 = \frac{1}{6}, B_3 = 0$   
 $\sum_{j=0}^m \binom{m+1}{j} B_j = 0$ , 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  
 $\prod_{n=1}^{\infty} (1 - x^n) = 1 + \sum_{k=1}^{\infty} (-1)^k (x^{k(3k+1)/2} + x^{k(3k-1)/2})$
- Catalan numbers  
 $C_n^{(k)} = \frac{1}{(k-1)n+1} \binom{kn}{n}$   
 $C^{(k)}(x) = 1 + x[C^{(k)}(x)]^k$
- Eulerian numbers  
Number of permutations  $\pi \in S_n$  in which exactly  $k$  elements are greater than the previous element.  $k$  j's s.t.  $\pi(j) > \pi(j+1)$ ,  $k+1$  j's s.t.  $\pi(j) \geq j$ ,  $k$  j's s.t.  $\pi(j) > j$ .  
 $E(n, k) = (n-k)E(n-1, k-1) + (k+1)E(n-1, k)$   
 $E(n, 0) = E(n, n-1) = 1$   
 $E(n, k) = \sum_{j=0}^k (-1)^j \binom{n+1}{j} (k+1-j)^n$

## 6.21 Tips for Generating Functions

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

## 7 Polynomial

### 7.1 Fast Fourier Transform

```
template<int MAXN>
struct FFT {
    using val_t = complex<double>;
    const double PI = acos(-1);
    val_t w[MAXN];
    FFT() {
        for (int i = 0; i < MAXN; ++i) {
            double arg = 2 * PI * i / MAXN;
            w[i] = val_t(cos(arg), sin(arg));
        }
    }
    void bitrev(val_t *a, int n); // see NTT
    void trans
        (val_t *a, int n, bool inv = false); // see NTT;
        // remember to replace LL with val_t
};
```

### 7.2 Number Theory Transform\*

```
//(2^16)+1, 65537, 3
//7*17*(2^23)+1, 998244353, 3
//1255*(2^20)+1, 1315962881, 3
//51*(2^25)+1, 1711276033, 29
template<int MAXN, ll P, ll RT> //MAXN must be 2^k
struct NTT {
    ll w[MAXN];
    ll mpow(ll a, ll n);
    ll minv(ll a) { return mpow(a, P - 2); }
    NTT() {
        ll dw = mpow(RT, (P - 1) / MAXN);
        w[0] = 1;
        for (int i = 1; i < MAXN; ++i) w[i] = w[i - 1] * dw % P;
    }
    void bitrev(ll *a, int n) {
        int i = 0;
        for (int j = 1; j < n - 1; ++j) {
            for (int k = n >> 1; (i ^= k) < k; k >>= 1);
```



```

    if (j < i) swap(a[i], a[j]);
}
}
void operator()(
    ll *a, int n, bool inv = false) { //0 <= a[i] < P
    bitrev(a, n);
    for (int L = 2; L <= n; L <= 1) {
        int dx = MAXN / L, dl = L >> 1;
        for (int i = 0; i < n; i += L) {
            for (int
                j = i, x = 0; j < i + dl; ++j, x += dx) {
                ll tmp = a[j + dl] * w[x] % P;
                if ((a[j]
                    + dl] = a[j] - tmp) < 0) a[j + dl] += P;
                if ((a[j] += tmp) >= P) a[j] -= P;
            }
        }
    }
    if (inv) {
        reverse(a + 1, a + n);
        ll invn = minv(n);
        for (int
            i = 0; i < n; ++i) a[i] = a[i] * invn % P;
    }
}
};

```

### 7.3 Fast Walsh Transform\*

```

/* 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];
}

```

### 7.4 Polynomial Operation

```

#define
    fi(s, n) for (int i = (int)(s); i < (int)(n); ++i)
template<int MAXN, ll P, ll RT> // MAXN = 2^k
struct Poly : vector<ll> { // coefficients in [0, P)
    using vector<ll>::vector;
    static NTT<MAXN, P, RT> ntt;
    int n() const { return (int)size(); } // n() >= 1
    Poly(const Poly &p, int m) : vector<ll>(m) {
        copy_n(p.data(), min(p.n(), m), data());
    }
    Poly& irev()
        { return reverse(data(), data() + n()), *this; }
    Poly& isz(int m) { return resize(m), *this; }
    Poly& iadd(const Poly &rhs) { // n() == rhs.n()
        fi(0, n()) if
            (((*this)[i] += rhs[i]) >= P) (*this)[i] -= P;
        return *this;
    }
    Poly& imul(ll k) {
        fi(0, n()) (*this)[i] = (*this)[i] * k % P;
        return *this;
    }
}

```

```

Poly Mul(const Poly &rhs) const {
    int m = 1;
    while (m < n() + rhs.n() - 1) m <= 1;
    Poly X(*this, m), Y(rhs, m);
    ntt(X.data(), m), ntt(Y.data(), m);
    fi(0, m) X[i] = X[i] * Y[i] % P;
    ntt(X.data(), m, true);
    return X.isz(n() + rhs.n() - 1);
}
Poly Inv() const { // (*this)[0] != 0, 1e5/95ms
    if (n() == 1) return {ntt.minv((*this)[0])};
    int m = 1;
    while (m < n() * 2) m <= 1;
    Poly Xi = Poly(*this, (n() + 1) / 2).Inv().isz(m);
    Poly Y(*this, m);
    ntt(Xi.data(), m), ntt(Y.data(), m);
    fi(0, m) {
        Xi[i] *= (2 - Xi[i] * Y[i]) % P;
        if ((Xi[i] %= P) < 0) Xi[i] += P;
    }
    ntt(Xi.data(), m, true);
    return Xi.isz(n());
}
Poly Sqrt()
    const { // Jacobi((*this)[0], P) = 1, 1e5/235ms
    if (n()
        == 1) return {QuadraticResidue((*this)[0], P)};
    Poly
        X = Poly(*this, (n() + 1) / 2).Sqrt().isz(n());
    return
        X.iadd(Mul(X.Inv()).isz(n())).imul(P / 2 + 1);
}
pair<Poly, Poly> DivMod
    (const Poly &rhs) const { // (rhs).back() != 0
    if (n() < rhs.n()) return {{0}, *this};
    const int m = n() - rhs.n() + 1;
    Poly X(rhs); X.irev().isz(m);
    Poly Y(*this); Y.irev().isz(m);
    Poly Q = Y.Mul(X.Inv()).isz(m).irev();
    X = rhs.Mul(Q), Y = *this;
    fi(0, n()) if ((Y[i] -= X[i]) < 0) Y[i] += P;
    return {Q, Y.isz(max(1, rhs.n() - 1))};
}
Poly Dx() const {
    Poly ret(n() - 1);
    fi(0,
        ret.n()) ret[i] = (i + 1) * (*this)[i + 1] % P;
    return ret.isz(max(1, ret.n()));
}
Poly Sx() const {
    Poly ret(n() + 1);
    fi(0, n())
        ret[i + 1] = ntt.minv(i + 1) * (*this)[i] % P;
    return ret;
}
Poly _tmul(int nn, const Poly &rhs) const {
    Poly Y = Mul(rhs).isz(n() + nn - 1);
    return Poly(Y.data() + n() - 1, Y.data() + Y.n());
}
vector<ll> _eval(const
    vector<ll> &x, const vector<Poly> &up) const {
    const int m = (int)x.size();
    if (!m) return {};
    vector<Poly> down(m * 2);
    // down[1] = DivMod(up[1]).second;
    // fi(2, m *
        2) down[i] = down[i / 2].DivMod(up[i]).second;
    down[1] = Poly(up[1])
        .irev().isz(n()).Inv().irev()._tmul(m, *this);
    fi(2, m * 2) down[i]
        = up[i ^ 1]._tmul(up[i].n() - 1, down[i / 2]);
    vector<ll> y(m);
    fi(0, m) y[i] = down[m + i][0];
    return y;
}
static vector<Poly> _tree1(const vector<ll> &x) {
    const int m = (int)x.size();
    vector<Poly> up(m * 2);
    fi(0, m) up[m + i] = {(x[i] ? P - x[i] : 0), 1};
    for (int i = m - 1; i
        > 0; --i) up[i] = up[i * 2].Mul(up[i * 2 + 1]);
    return up;
}
vector
    <ll> Eval(const vector<ll> &x) const { // 1e5, 1s
    auto up = _tree1(x); return _eval(x, up);
}

```

```

}
static Poly Interpolate(const vector
<ll> &x, const vector<ll> &y) { // 1e5, 1.4s
    const int m = (int)x.size();
    vector<Poly> up = _tree1(x), down(m * 2);
    vector<ll> z = up[1].Dx()._eval(x, up);
    fi(0, m) z[i] = y[i] * ntt.minv(z[i]) % P;
    fi(0, m) down[m + i] = {z[i]};
    for (int i = m - 1; i > 0; --i) down[i] = down[i * 2].Mul(up[i
* 2 + 1]).iadd(down[i * 2 + 1].Mul(up[i * 2]));
    return down[1];
}
Poly Ln() const { // (*this)[0] == 1, 1e5/170ms
    return Dx().Mul(Inv()).Sx().isz(n());
}
Poly Exp() const { // (*this)[0] == 0, 1e5/360ms
    if (n() == 1) return {1};
    Poly X = Poly(*this, (n() + 1) / 2).Exp().isz(n());
    Poly Y = X.Ln(); Y[0] = P - 1;
    fi(0, n())
        if ((Y[i] = (*this)[i] - Y[i]) < 0) Y[i] += P;
    return X.Mul(Y).isz(n());
}
// M := P(P - 1). If k >= M, k := k % M + M.
Poly Pow(ll k) const {
    int nz = 0;
    while (nz < n() && !(*this)[nz]) ++nz;
    if (nz * min(k, (ll)n()) >= n()) return Poly(n());
    if (!k) return Poly(Poly{1}, n());
    Poly X(data() + nz, data() + nz + n() - nz * k);
    const ll c = ntt.mpow(X[0], k % (P - 1));
    return X.Ln().imul
        (k % P).Exp().imul(c).irev().isz(n()).irev();
}
static ll
    LinearRecursion(const vector<ll> &a, const vector
<ll> &coef, ll n) { // a_n = \sum c_j a_{n-j}
    const int k = (int)a.size();
    assert((int)coef.size() == k + 1);
    Poly C(k + 1), W(Poly{1}, k), M = {0, 1};
    fi(1, k + 1) C[k - i] = coef[i] ? P - coef[i] : 0;
    C[k] = 1;
    while (n) {
        if (n % 2) W = W.Mul(M).DivMod(C).second;
        n /= 2, M = M.Mul(M).DivMod(C).second;
    }
    ll ret = 0;
    fi(0, k) ret = (ret + W[i] * a[i]) % P;
    return ret;
}
};
#undef fi
using Poly_t = Poly<131072 * 2, 998244353, 3>;
template<> decltype(Poly_t::ntt) Poly_t::ntt = {};

```

## 7.5 Value Polynomial

```

struct Poly {
    mint base; // f(x) = poly[x - base]
    vector<mint> poly;
    Poly(mint b = 0, mint x = 0): base(b), poly(1, x) {}
    mint get_val(const mint &x) {
        if (x >= base && x < base + SZ(poly))
            return poly[x - base];
        mint rt = 0;
        vector<mint> lmul(SZ(poly), 1), rmul(SZ(poly), 1);
        for (int i = 1; i < SZ(poly); ++i)
            lmul[i] = lmul[i - 1] * (x - (base + i - 1));
        for (int i = SZ(poly) - 2; i >= 0; --i)
            rmul[i] = rmul[i + 1] * (x - (base + i + 1));
        for (int i = 0; i < SZ(poly); ++i)
            rt += poly[i] * ifac[i] * inegfac
                [SZ(poly) - 1 - i] * lmul[i] * rmul[i];
        return rt;
    }
    void raise() { // g(x) = sigma{base:x} f(x)
        if (SZ(poly) == 1 && poly[0] == 0)
            return;
        mint nw = get_val(base + SZ(poly));
        poly.pb(nw);
        for (int i = 1; i < SZ(poly); ++i)
            poly[i] += poly[i - 1];
    }
};

```

## 7.6 Newton's Method

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

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

## 8 Geometry

### 8.1 Default Code

```

typedef pair<double, double> pdd;
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.X + b.X, a.Y + b.Y); }
pdd operator-(pdd a, pdd b)
{ return pdd(a.X - b.X, a.Y - b.Y); }
pdd operator*(pdd a, double b)
{ return pdd(a.X * b, a.Y * b); }
pdd operator/(pdd a, double b)
{ return pdd(a.X / b, a.Y / b); }
double dot(pdd a, pdd b)
{ return a.X * b.X + a.Y * b.Y; }
double cross(pdd a, pdd b)
{ return a.X * b.Y - a.Y * b.X; }
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.Y, p1.X); }
pdd projection(pdd p1, pdd p2, pdd p3)
{ return p1 + (
    p2 - p1) * dot(p3 - p1, p2 - p1) / abs2(p2 - p1); }
pdd reflection(pdd p1, pdd p2, pdd p3)
{ return p3 + perp(p2 - p1
    ) * cross(p3 - p1, p2 - p1) / abs2(p2 - p1) * 2; }
pdd linearTransformation
    (pdd p0, pdd p1, pdd q0, pdd q1, pdd r) {
    pdd dp = p1 - p0
        , dq = q1 - q0, num(cross(dp, dq), dot(dp, dq));
    return q0 + pdd(
        cross(r - p0, num), dot(r - p0, num)) / abs2(dp);
} // from line p0--p1 to q0--q1, apply to r

```

### 8.2 PointSegDist\*

```

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));
}

```

### 8.3 Heart

```
pdd circenter
    (pdd p0, pdd p1, pdd p2) { // radius = abs(center)
    p1 = p1 - p0, p2 = p2 - p0;
    double x1 = p1.X, y1 = p1.Y, x2 = p2.X, y2 = p2.Y;
    double m = 2. * (x1 * y2 - y1 * x2);
    center.X = (x1 * x1
        * y2 - x2 * x2 * y1 + y1 * y2 * (y1 - y2)) / m;
    center.Y = (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; }
```

### 8.4 point in circle

```
// return
    p4 is strictly in circumcircle of tri(p1,p2,p3)
ll sqr(ll x) { return x * x; }
bool in_cc(const pll&
    p1, const pll& p2, const pll& p3, const pll& p4) {
    ll u11 = p1.X - p4.X; ll u12 = p1.Y - p4.Y;
    ll u21 = p2.X - p4.X; ll u22 = p2.Y - p4.Y;
    ll u31 = p3.X - p4.X; ll u32 = p3.Y - p4.Y;
    ll u13
        = sqr(p1.X) - sqr(p4.X) + sqr(p1.Y) - sqr(p4.Y);
    ll u23
        = sqr(p2.X) - sqr(p4.X) + sqr(p2.Y) - sqr(p4.Y);
    ll u33
        = sqr(p3.X) - sqr(p4.X) + sqr(p3.Y) - sqr(p4.Y);
    __int128 det = (__int128)-u13 * u22 * u31
        + (__int128)u12 * u23 * u31 + (__int128)u13 *
        u21 * u32 - (__int128)u11 * u23 * u32 - (__int128)
        u12 * u21 * u33 + (__int128)u11 * u22 * u33;
    return det > eps;
}
```

### 8.5 Convex hull\*

```
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 = SZ(ans); i < SZ(dots); ans.pb(dots[i++]))
            while (SZ(ans) > t && ori
                (ans[SZ(ans) - 2], ans.back(), dots[i]) <= 0)
                ans.pop_back();
    ans.pop_back(), ans.swap(dots);
}
```

### 8.6 PointInConvex\*

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

### 8.7 TangentPointToHull\*

```
/* The point should be strictly out of hull
    return arbitrary point on the tangent line */
pii get_tangent(vector<pll> &C, pll p) {
    auto gao = [&](int s) {
        return cyc_tsearch(SZ(C), [&](int x, int y)
            { return ori(p, C[x], C[y]) == s; });
    };
    return pii(gao(1), gao(-1));
} // return (a, b), ori(p, C[a], C[b]) >= 0
```

### 8.8 Intersection of line and convex

```
int TangentDir(vector<pll> &C, pll dir) {
    return cyc_tsearch(SZ(C), [&](int a, int b) {
        return cross(dir, C[a]) > cross(dir, C[b]);
    });
}
#define cml(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 = SZ(C);
    if (cml(A) < 0 || cml(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;
            (cml(m) == cml(t) ? l : r) = m;
        }
        return (l + !cml(r)) % n;
    };
    pii res = pii(gao(B, A), gao(A, B)); // (i, j)
    if (res.X == res.Y) // touching the corner i
        return pii(res.X, -1);
    if (!
        cml(res.X) && !cml(res.Y)) // along side i, i+1
        switch ((res.X - res.Y + n + 1) % n) {
            case 0: return pii(res.X, res.X);
            case 2: return pii(res.Y, res.Y);
        }
    /* 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]
```

### 8.9 minMaxEnclosingRectangle\*

```
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 = SZ(dots);
    dots.pb(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);
}
```

### 8.10 VectorInPoly\*

```
// ori(a
    , b, c) >= 0, valid: "strict" angle from a-b to a-c
bool btwangle(pll a, pll b, pll c, pll p, int strict) {
    return
        ori(a, b, p) >= strict && ori(a, p, c) >= strict;
}
```

```

}
// whether vector
{cur, p} in counter-clockwise order prv, cur, nxt
bool inside
(pll prv, pll cur, pll nxt, pll p, int strict) {
    if (ori(cur, nxt, prv) >= 0)
        return btwangle(cur, nxt, prv, p, strict);
    return !btwangle(cur, prv, nxt, p, !strict);
}

```

## 8.11 PolyUnion\*

```

double rat(pll a, pll b) {
    return sign
        (b.X) ? (double)a.X / b.X : (double)a.Y / b.Y;
} // all poly. should be ccw
double polyUnion(vector<vector<pll>> &poly) {
    double res = 0;
    for (auto &p : poly)
        for (int a = 0; a < SZ(p); ++a) {
            pll A = p[a], B = p[(a + 1) % SZ(p)];
            vector
                <pair<double, int>> segs = {{0, 0}, {1, 0}};
            for (auto &q : poly) {
                if (&p == &q) continue;
                for (int b = 0; b < SZ(q); ++b) {
                    pll C = q[b], D = q[(b + 1) % SZ(q)];
                    int sc = ori(A, B, C), sd = ori(A, B, D);
                    if (sc != sd && min(sc, sd) < 0) {
                        double sa = cross(D
                            - C, A - C), sb = cross(D - C, B - C);
                        segs.emplace_back
                            (sa / (sa - sb), sign(sc - sd));
                    }
                    if (!sc && !sd &&
                        &q < &p && sign(dot(B - A, D - C)) > 0) {
                        segs.emplace_back(rat(C - A, B - A), 1);
                        segs.emplace_back(rat(D - A, B - A), -1);
                    }
                }
            }
            sort(ALL(segs));
            for (auto &s : segs) s.X = clamp(s.X, 0.0, 1.0);
            double sum = 0;
            int cnt = segs[0].second;
            for (int j = 1; j < SZ(segs); ++j) {
                if (!cnt) sum += segs[j].X - segs[j - 1].X;
                cnt += segs[j].Y;
            }
            res += cross(A, B) * sum;
        }
    return res / 2;
}

```

## 8.12 Polar Angle Sort\*

```

int cmp(pll a, pll b, bool same = true) {
#define is_neg(k) (
    sign(k.Y) < 0 || (sign(k.Y) == 0 && sign(k.X) < 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;
}

```

## 8.13 Half plane intersection\*

```

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

```

```

return ori(a.X, a.Y, b.Y) < 0;
});
deque<Line> dq(1, arr[0]);
for (auto p : arr) {
    if (cmp(
        dq.back().Y - dq.back().X, p.Y - p.X, 0) == -1)
        continue;
    while (SZ(dq)
        ) >= 2 && !isin(p, dq[SZ(dq) - 2], dq.back())
        dq.pop_back();
    while (SZ(dq) >= 2 && !isin(p, dq[0], dq[1]))
        dq.pop_front();
    dq.pb(p);
}
while (SZ(dq)
    ) >= 3 && !isin(dq[0], dq[SZ(dq) - 2], dq.back())
    dq.pop_back();
while (SZ(dq) >= 3 && !isin(dq.back(), dq[0], dq[1]))
    dq.pop_front();
return vector<Line>(ALL(dq));
}

```

## 8.14 RotatingSweepLine

```

void rotatingSweepLine(vector<pii> &ps) {
    int n = SZ(ps), 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.Y] - ps[a.X], ps[b.Y] - ps[b.X]);
    }); // cmp(): polar angle compare
    iota(ALL(id), 0);
    sort(ALL(id), [&](int a, int b) {
        if (ps[a].Y != ps[b].Y) return ps[a].Y < ps[b].Y;
        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.X], pos[l.Y], id[pos[l.X]], id[pos[l.Y]
            ]) = make_tuple(pos[l.Y], pos[l.X], l.Y, l.X);
    }
}

```

## 8.15 Minimum Enclosing Circle\*

```

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 < SZ(dots); ++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;
}

```

## 8.16 Intersection of two circles\*

```

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

```

## 8.17 Intersection of polygon and circle\*

```
// 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 &0,const double r){
    double S=0;
    for(int i=0;i<SZ(poly);++i)
        S+=_area(poly[i]-0,poly[(i+1)%SZ(poly)
            ]-0,r)*ori(0,poly[i],poly[(i+1)%SZ(poly)]);
    return fabs(S);
}
```

## 8.18 Intersection of line and circle\*

```
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};
}
```

## 8.19 Tangent line of two circles

```
vector<Line>
    > go( const Cir& c1 , const Cir& c2 , int sign1 ){
    // sign1 = 1 for outer tang, -1 for inner 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.X * c - sign2 * h * v.Y,
            v.Y * c + sign2 * h * v.X);
        pdd p1 = c1.0 + n * c1.R;
        pdd p2 = c2.0 + n * (c2.R * sign1);
        if (sign(p1.X - p2.X) == 0 and
            sign(p1.Y - p2.Y) == 0)
            p2 = p1 + perp(c2.0 - c1.0);
        ret.pb(Line(p1, p2));
    }
    return ret;
}
```

## 8.20 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 disjunct(Cir &a, Cir &b, int x)
    {return sign(abs(a.0 - b.0) - a.R - b.R) > x;}
    bool contain(Cir &a, Cir &b, int x)
    {return sign(a.R - b.R - abs(a.0 - b.0)) > 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] ||
                    disjunct(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].0.Y, aa.X - c[i].0.X);
                    double B =
                        atan2(bb.Y - c[i].0.Y, bb.X - c[i].0.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;
                }
            }
        }
    }
};
```

```
bool operator<(const Teve &a)const
    {return ang < a.ang;}
eve[N * 2];
// strict: x = 0, otherwise x = -1
bool disjunct(Cir &a, Cir &b, int x)
{return sign(abs(a.0 - b.0) - a.R - b.R) > x;}
bool contain(Cir &a, Cir &b, int x)
{return sign(a.R - b.R - abs(a.0 - b.0)) > 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] ||
                disjunct(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].0.Y, aa.X - c[i].0.X);
                double B =
                    atan2(bb.Y - c[i].0.Y, bb.X - c[i].0.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;
            }
        }
    }
};
```

## 8.21 3Dpoint\*

```
struct Point {
    double x, y, z;
    Point(double _x = 0, double
        _y = 0, double _z = 0): x(_x), y(_y), z(_z){}
    Point(pdd p) { x = p.X, y = p.Y, z = abs2(p); }
};
Point operator-(const Point &p1, const Point &p2)
{ return
    Point(p1.x - p2.x, p1.y - p2.y, p1.z - p2.z); }
Point operator+(const Point &p1, const Point &p2)
{ return
    Point(p1.x + p2.x, p1.y + p2.y, p1.z + p2.z); }
Point operator/(const Point &p1, const double &v)
{ return Point(p1.x / v, p1.y / v, p1.z / v); }
Point cross(const Point &p1, const Point &p2)
{ return Point(p1.y * p2.z - p1.z * p2.y, p1.z
    * p2.x - p1.x * p2.z, p1.x * p2.y - p1.y * p2.x); }
double dot(const Point &p1, const Point &p2)
{ return p1.x * p2.x + p1.y * p2.y + p1.z * p2.z; }
double abs(const Point &a)
{ return sqrt(dot(a, a)); }
Point cross3
    (const Point &a, const Point &b, const Point &c)
{ return cross(b - a, c - a); }
double area(Point a, Point b, Point c)
{ return abs(cross3(a, b, c)); }
```



```
double volume(Point a, Point b, Point c, Point d)
{ return dot(cross3(a, b, c), d - a); }
Point masscenter(Point a, Point b, Point c, Point d)
{ return (a + b + c + d) / 4; }
pdd proj(Point a, Point b, Point c, Point u) {
// proj. u to the plane of a, b, and c
Point e1 = b - a;
Point e2 = c - a;
e1 = e1 / abs(e1);
e2 = e2 - e1 * dot(e2, e1);
e2 = e2 / abs(e2);
Point p = u - a;
return pdd(dot(p, e1), dot(p, e2));
}
```

## 8.22 Convexhull3D\*

```
struct convex_hull_3D {
struct Face {
int a, b, c;
Face(int ta, int tb, int tc): a(ta), b(tb), c(tc) {}
}; // return the faces with pt indexes
vector<Face> res;
vector<Point> P;
convex_hull_3D(const vector<Point> &_P): res(), P(_P) {
// all points coplanar case will WA, O(n^2)
int n = SZ(P);
if (n <= 2) return; // be careful about edge case
// ensure first 4 points are not coplanar
swap(P[1], *find_if(ALL(P), [&](auto p) { return
sign(abs2(P[0] - p)) != 0; }));
swap(P[2], *find_if(ALL(P), [&](auto p) { return
sign(abs2(cross3(p, P[0], P[1]))) != 0; }));
swap(P[3], *find_if(ALL(P), [&](auto p) { return
sign(volume(P[0], P[1], P[2], p)) != 0; }));
vector<vector<int>> flag(n, vector<int>(n));
res.emplace_back(0, 1, 2); res.emplace_back(2, 1, 0);
for (int i = 3; i < n; ++i) {
vector<Face> next;
for (auto f : res) {
int d
= sign(volume(P[f.a], P[f.b], P[f.c], P[i]));
if (d <= 0) next.pb(f);
int ff = (d > 0) - (d < 0);
flag[f.a][f.b] = flag[f.b][f.c] = flag[f.c][f.a] = ff;
}
for (auto f : res) {
auto F = [&](int x, int y) {
if (flag[x][y] > 0 && flag[y][x] <= 0)
next.emplace_back(x, y, i);
};
F(f.a, f.b); F(f.b, f.c); F(f.c, f.a);
}
res = next;
}
}
bool same(Face s, Face t) {
if (sign(volume(P[s.a], P[s.b], P[s.c], P[t.a])) != 0) return 0;
if (sign(volume(P[s.a], P[s.b], P[s.c], P[t.b])) != 0) return 0;
if (sign(volume(P[s.a], P[s.b], P[s.c], P[t.c])) != 0) return 0;
return 1;
}
int polygon_face_num() {
int ans = 0;
for (int i = 0; i < SZ(res); ++i)
ans += none_of(res.begin(), res.begin()
+ i, [&](Face g) { return same(res[i], g); });
return ans;
}
double get_volume() {
double ans = 0;
for (auto f : res)
ans += volume(Point(0, 0, 0), P[f.a], P[f.b], P[f.c]);
return fabs(ans / 6);
}
double get_dis(Point p, Face f) {
Point p1 = P[f.a], p2 = P[f.b], p3 = P[f.c];
double a = (p2.y - p1.y) * (p3.z - p1.z) - (p2.z - p1.z) * (p3.y - p1.y);
double b = (p2.z - p1.z) * (p3.x - p1.x) - (p3.z - p1.z) * (p2.x - p1.x);
double c = (p2.x - p1.x) * (p3.y - p1.y) - (p2.y - p1.y) * (p3.x - p1.x);
double d = 0 - (a * p1.x + b * p1.y + c * p1.z);
return fabs(a * p.x + b * p.y + c * p.z + d) / sqrt(a * a + b * b + c * c);
}
```

```
double c = (p2.x - p1.x) * (p3.y - p1.y) - (p2.y - p1.y) * (p3.x - p1.x);
double d = 0 - (a * p1.x + b * p1.y + c * p1.z);
return fabs(a * p.x + b * p.y + c * p.z + d) / sqrt(a * a + b * b + c * c);
}
```

## 8.23 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(trl, 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.pb(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);
}
}

```

## 8.24 Triangulation Voronoi\*

```

vector<Line> ls[N];
pll arr[N];
Line make_line(pdd p, Line l) {
    pdd d = l.Y - l.X; d = perp(d);
    pdd m = (l.X + l.Y) / 2;
    l = Line(m, m + d);
    if (ori(l.X, l.Y, p) < 0)
        l = Line(m + d, m);
    return l;
}
double calc_area(int id) {
    // use to calculate
    // the area of point "strictly in the convex hull"
    vector<Line> hpi = halfPlaneInter(ls[id]);
    vector<pdd> ps;
    for (int i = 0; i < SZ(hpi); ++i)
        ps.pb(intersect(hpi[i].X, hpi[i].Y, hpi[(i
            + 1) % SZ(hpi)].X, hpi[(i + 1) % SZ(hpi)].Y));
    double rt = 0;
    for (int i = 0; i < SZ(ps); ++i)
        rt += cross(ps[i], ps[(i + 1) % SZ(ps)]);
    return fabs(rt) / 2;
}
void solve(int n, pii *oarr) {
    map<pll, int> mp;
    for (int i = 0; i < n; ++i)
        arr[i] = pll(oarr[i].X, oarr[i].Y), mp[arr[i]] = i;
    build(n, arr); // Triangulation
    for (auto *t : triang) {
        vector<int> p;
        for (int i = 0; i < 3; ++i)
            if (mp.find(t->p[i]) != mp.end())
                p.pb(mp[t->p[i]]);
    }
}

```

```

    for (int i = 0; i < SZ(p); ++i)
        for (int j = i + 1; j < SZ(p); ++j) {
            Line l(oarr[p[i]], oarr[p[j]]);
            ls[p[i]].pb(make_line(oarr[p[i]], l));
            ls[p[j]].pb(make_line(oarr[p[j]], l));
        }
}
}

```

## 8.25 Minkowski Sum\*

```

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

```

## 9 Else

### 9.1 Cyclic Ternary Search\*

```

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

```

### 9.2 Mo's Alogrithm(With modification)

```

/*
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
    }
}

```

### 9.3 Mo's Alogrithm On Tree

```

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

```

```

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

```

## 9.4 Additional Mo's Algorithm Trick

- 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  $l/blk \neq curL/blk$ , 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:  $f([l, r], r+1) = f([1, r], r+1) - f([1, l], r+1)$ .
  - Part1: Answer all  $f([1, r], r+1)$  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.

## 9.5 Hilbert Curve

```

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

```

## 9.6 DynamicConvexTrick\*

```

// only works for integer coordinates!! maintain max
struct Line {
    mutable ll a, b, p;
    bool operator<(const Line &rhs) const { return a < rhs.a; }
    bool operator<(ll x) const { return p < x; }
};
struct DynamicHull : multiset<Line, less<>> {
    static const ll kInf = 1e18;
    ll Div(ll a,
        ll b) { return a / b - ((a ^ b) < 0 && a % b); }
    bool isect(iterator x, iterator y) {
        if (y == end()) { x->p = kInf; return 0; }
        if (x
            ->a == y->a) x->p = x->b > y->b ? kInf : -kInf;
        else x->p = Div(y->b - x->b, x->a - y->a);
        return x->p >= y->p;
    }
    void addline(ll a, ll b) {
        auto z = insert({a, b, 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) {
    auto l = *lower_bound(x);
    return l.a * x + l.b;
}
};

```

## 9.7 All LCS\*

```

void all_lcs(string s, string t) { // 0-base
    vector<int> h(SZ(t));
    iota(ALL(h), 0);
    for (int a = 0; a < SZ(s); ++a) {
        int v = -1;
        for (int c = 0; c < SZ(t); ++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 !!
    }
}

```

## 9.8 DLX\*

```

#define TRAV(i, link, start)
    for (int i = link[start]; i != start; i = link[i])
template<bool A, bool B = !A> // A: Exact
struct DLX {
    int lt[NN], rg[NN], up[NN], dn[NN]
        , cl[NN], rw[NN], bt[NN], s[NN], head, sz, ans;
    int columns;
    bool vis[NN];
    void remove(int c) {
        if (A) lt[rg[c]] = lt[c], rg[lt[c]] = rg[c];
        TRAV(i, dn, c) {
            if (A) {
                TRAV(j, rg, i)
                    up[dn[j]]
                        = up[j], dn[up[j]] = dn[j], --s[cl[j]];
            } else {
                lt[rg[i]] = lt[i], rg[lt[i]] = rg[i];
            }
        }
    }
    void restore(int c) {
        TRAV(i, up, c) {
            if (A) {
                TRAV(j, lt, i)
                    ++s[cl[j]], up[dn[j]] = j, dn[up[j]] = j;
            } else {
                lt[rg[i]] = rg[lt[i]] = i;
            }
        }
        if (A) lt[rg[c]] = c, rg[lt[c]] = c;
    }
    void init(int c) {
        columns = c;
        for (int i = 0; i < c; ++i) {
            up[i] = dn[i] = bt[i] = i;
            lt[i] = i == 0 ? c : i - 1;
            rg[i] = i == c - 1 ? c : i + 1;
            s[i] = 0;
        }
        rg[c] = 0, lt[c] = c - 1;
        up[c] = dn[c] = -1;
        head = c, sz = c + 1;
    }
    void insert(int r, const vector<int> &col) {
        if (col.empty()) return;
        int f = sz;
        for (int i = 0; i < (int)col.size(); ++i) {
            int c = col[i], v = sz++;
            dn[bt[c]] = v;
            up[v] = bt[c], bt[c] = v;
            rg[v] = (i + 1 == (int)col.size() ? f : v + 1);
            rw[v] = r, cl[v] = c;
            ++s[c];
            if (i > 0) lt[v] = v - 1;
        }
        lt[f] = sz - 1;
    }
}

```

```

}
int h() {
    int ret = 0;
    memset(vis, 0, sizeof(bool) * sz);
    TRAV(x, rg, head) {
        if (vis[x]) continue;
        vis[x] = true, ++ret;
        TRAV(i, dn, x) TRAV(j, rg, i) vis[cl[j]] = true;
    }
    return ret;
}
void dfs(int dep) {
    if (dep + (A ? 0 : h()) >= ans) return;
    if (rg[head] == head) return ans = dep, void();
    if (dn[rg[head]] == rg[head]) return;
    int w = rg[head];
    TRAV(x, rg, head) if (s[x] < s[w]) w = x;
    if (A) remove(w);
    TRAV(i, dn, w) {
        if (B) remove(i);
        TRAV(j, rg, i) remove(A ? cl[j] : j);
        dfs(dep + 1);
        TRAV(j, lt, i) restore(A ? cl[j] : j);
        if (B) restore(i);
    }
    if (A) restore(w);
}
int solve() {
    for (int i = 0; i < columns; ++i)
        dn[bt[i]] = i, up[i] = bt[i];
    ans = 1e9, dfs(0);
    return ans;
}
};

```

## 9.9 Matroid Intersection

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

- $Y_1 = \{x \notin S \mid S \cup \{x\} \in I_1\}$
- $Y_2 = \{x \notin S \mid S \cup \{x\} \in I_2\}$

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

- $x \rightarrow y$  if  $S - \{x\} \cup \{y\} \in I_1$ .
- $y \rightarrow x$  if  $S - \{x\} \cup \{y\} \in I_2$ .

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

## 9.10 AdaptiveSimpson

```

using F_t = function<double(double)>;
pdd simpson(const F_t &f, double l, double r,
            double fl, double fr, double fm = nan("")) {
    if (isnan(fm)) fm = f((l + r) / 2);
    return {fm, (r - l) / 6 * (fl + 4 * fm + fr)};
}
double simpson_ada(const F_t &f, double l, double r,
                  double fl, double fm, double fr, double eps) {
    double m = (l + r) / 2,
           s = simpson(f, l, r, fl, fr, fm).second;
    auto [flm, sl] = simpson(f, l, m, fl, fm);
    auto [fmr, sr] = simpson(f, m, r, fm, fr);
    double delta = sl + sr - s;
    if (abs(delta) <= 15 * eps)
        return sl + sr + delta / 15;
    return simpson_ada(f, l, m, fl, flm, fm, eps / 2) +
           simpson_ada(f, m, r, fm, fmr, fr, eps / 2);
}
double simpson_ada(const F_t &f, double l, double r) {
    return simpson_ada(f, l, r, f(l), f((l + r) / 2), f(r), 1e-9 / 7122);
}
double simpson_ada2(const F_t &f, double l, double r) {
    double h = (r - l) / 7122, s = 0;
    for (int i = 0; i < 7122; ++i, l += h)
        s += simpson_ada(f, l, l + h);
    return s;
}

```

## 9.11 Simulated Annealing

```

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;
}

```

## 9.12 Tree Hash\*

```

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

```

## 9.13 Binary Search On Fraction

```

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

```

## 10 Python

### 10.1 Misc

```

from decimal import *
setcontext(Context(prec
                  =MAX_PREC, Emax=MAX_EMAX, rounding=ROUND_FLOOR))
print(Decimal(input()) * Decimal(input()))
from fractions import Fraction
Fraction(
    ('3.14159').limit_denominator(10).numerator # 22

```