

Contents

1 Basic	1		
1.1 Default code	1		
1.2 vimrc	1		
1.3 read	1		
1.4 Black Magic	1		
2 Graph	2		
2.1 BCC Vertex*	2		
2.2 Bridge*	2		
2.3 2SAT (SCC)*	2		
2.4 MinimumMeanCycle*	3		
2.5 Virtual Tree*	3		
2.6 Maximum Clique Dyn*	3		
2.7 Minimum Steiner Tree*	3		
2.8 Dominator Tree*	4		
2.9 Minimum Arborescence*	4		
2.10 Vizing's theorem	4		
2.11 Minimum Clique Cover*	5		
2.12 Number of Maximal Clique*	5		
2.13 Dijkstra	5		
2.14 Theory	6		
3 Data Structure	6		
3.1 Leftist Tree	6		
3.2 Heavy light Decomposition	6		
3.3 Centroid Decomposition*	6		
3.4 Treap	7		
3.5 Range Operations	7		
4 Flow/Matching	8		
4.1 Kuhn Munkres	8		
4.2 MincostMaxflow	8		
4.3 Maximum Simple Graph Matching*	9		
4.4 Minimum Weight Matching (Clique version)*	9		
4.5 SW-mincut	9		
4.6 BoundedFlow(Dinic*)	10		
4.7 Gomory Hu tree	10		
5 String	10		
5.1 KMP	10		
5.2 Z-value	11		
5.3 Manacher*	11		
5.4 Suffix Array	11		
5.5 SAIS*	11		
5.6 Aho-Corasick Automatan	12		
5.7 Smallest Rotation	12		
5.8 De Bruijn sequence*	12		
5.9 SAM	12		
5.10 cyclic LCS	13		
6 Math	13		
6.1 ax+by=gcd*	13		
6.2 floor and ceil	13		
6.3 floor sum*	13		
6.4 Miller Rabin*	13		
6.5 Big number	13		
6.6 Fraction	14		
6.7 Simultaneous Equations	15		
6.8 Pollard Rho	15		
6.9 Simplex Algorithm	15		
6.10 Schreier-Sims Algorithm*	15		
6.11 Chinese Remainder	16		
6.12 Quadratic Residue	16		
6.13 Pi Count	16		
6.14 Primes	16		
6.15 Prime Factorization	17		
6.16 Theorem	17		
6.16.1 Kirchhoff's Theorem	17		
6.16.2 Tutte's Matrix	17		
6.16.3 Cayley's Formula	17		
7 Polynomial	17		
7.1 Fast Fourier Transform	17		
7.2 Number Theory Transform	17		
7.3 Fast Walsh Transform*	17		
7.4 Polynomial Operation	18		
8 Geometry	19		
8.1 Default Code	19		
8.2 Convex hull*	19		
8.3 External bisector	19		
8.4 Heart	19		
8.5 Minimum Circle Cover*	19		
8.6 Polar Angle Sort*	20		
8.7 Intersection of two circles*	20		
8.8 Intersection of polygon and circle	20		
8.9 Intersection of line and circle	20		
8.10 point in circle	20		
8.11 Half plane intersection	20		
8.12 Circle Cover*	21		
8.13 3D point*	21		
8.14 Convex hull 3D*	21		
8.15 Delaunay Triangulation*	22		
8.16 Triangulation Voronoi*	23		
8.17 Tangent line of two circles	23		
8.18 minMaxEnclosingRectangle	23		
8.19 minDistOfTwoConvex	24		
8.20 Minkowski Sum*	24		
8.21 Rotating Sweep Line	24		
8.22 Simpson	24		
9 Else	24		
9.1 Mo's Algorithm (With modification)	24		
9.2 Mo's Algorithm On Tree	24		
9.3 Dynamic Convex Trick*	25		
9.4 Matroid Intersection	25		

1 Basic

1.1 Default code

```
#pragma GCC (2)
#include <bits/stdc++.h>
using namespace std;
#define f first
#define s second
#define sz(i) ((int)i.size())
#define all(i) i.begin(), i.end()
#define pb push_back
#define endl '\n'
int main(){
    ios::sync_with_stdio(false), cin.tie(0), cout.tie(0);
}
```

1.2 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
syn on
hi cursorline cterm=none ctermbg=89
inoremap {<ENTER> {<CR>}<Esc>ko<TAB>
map <F8> <Esc>:w<CR>:!g++ "%<" -std=c++17 -Wall
-Wextra -Wshadow -Wconversion -fsanitize=address,
undefined -g && echo success<CR>
map <F9> <Esc>:w<CR>:!g++ "%<" -std=c++17
&& echo success<CR>
map <F10> <Esc>:!. / "%<"<CR>
```

1.3 read

```
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++;
}
inline int readint(){
    int f=1, x=0; char c;
    do{c=getchar(); if(c=='-')f=-1;}while(c<'0' || c>'9');
    do{x=x*10+c-'0'; c=getchar();}while(c>='0' && c<='9');
    return f*x;
}
```

1.4 Black Magic

```
#include <ext/pb_ds/priority_queue.hpp>
#include <ext/pb_ds/assoc_container.hpp> //rb_tree
using namespace __gnu_pbds;
struct myhash {
    const int RANDOM = (1ll)(make_unique<char>()).get() ^
        chrono::high_resolution_clock::now().
        time_since_epoch().count();
    static unsigned ll hash_f(unsigned ll x) {
        x += 0x9e3779b97f4a7c15;
        x = (x ^ (x >> 30)) * 0xbf58476d1ce4e5b9;
        x = (x ^ (x >> 27)) * 0x94d049bb133111eb;
```

```

    return x ^ (x >> 31);
}
static unsigned hash_combine(unsigned a, unsigned b){
    return a * 31 + b; }
ll operator()(ll x) const{ return hash_f(x)^RANDOM; }
};
typedef gp_hash_table<ll, int, myhash> hashtable;
typedef __gnu_pbds::priority_queue<int> heap;
int main() {
    heap h1, h2;
    h1.push(1), h1.push(3);
    h2.push(2), h2.push(4);
    h1.join(h2);
    cout << h1.size() << h2.size() << h1.top() << endl;
    //404
    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
}
//__int128_t, __float128_t

```

2 Graph

2.1 BCC Vertex*

```

vector<int> G[N]; // 1-base
vector<int> nG[N], 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];
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) {
    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);
    // circle-square 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[i], low[u]);
            else if (instack[i] && dfn[i] < dfn[u])
                low[u] = min(low[u], dfn[i]);
        if (low[u] == dfn[u]) {
            int tmp;
            do {
                tmp = st.top(), st.pop();
                instack[tmp] = 0, bln[tmp] = nScc;
            } while (tmp != u);
            ++nScc;
        }
    }
    bool solve() {
        Time = nScc = 0;
        for (int i = 0; i < n + n; ++i)
            SCC[i].clear(), low[i] = dfn[i] = bln[i] = 0;
        for (int i = 0; i < n + n; ++i)
            if (!dfn[i]) dfs(i);
        for (int i = 0; i < n + n; ++i) SCC[bln[i]].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 < n; ++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*

```

const int N = 150;
struct MaxClique { // Maximum Clique
    bitset<N> a[N], cs[N];

```

```

int ans, sol[N], q, cur[N], d[N], n;
void init(int _n) {
    n = _n;
    for (int i = 0; i < n; i++) a[i].reset();
}
void addEdge(int u, int v) { a[u][v] = a[v][u] = 1; }
void csort(vector<int> &r, vector<int> &c) {
    int mx = 1, km = max(ans - q + 1, 1), t = 0,
        m = r.size();
    cs[1].reset(), cs[2].reset();
    for (int i = 0; i < m; i++) {
        int p = r[i], k = 1;
        while ((cs[k] & a[p]).count()) k++;
        if (k > mx) mx++, cs[mx + 1].reset();
        cs[k][p] = 1;
        if (k < km) r[t++] = p;
    }
    c.resize(m);
    if (t) c[t - 1] = 0;
    for (int k = km; k <= mx; k++)
        for (int p = cs[k]._Find_first(); p < N;
            p = cs[k]._Find_next(p))
            r[t] = p, c[t] = k, t++;
}
void dfs(vector<int> &r, vector<int> &c, int l,
    bitset<N> mask) {
    while (!r.empty()) {
        int p = r.back();
        r.pop_back(), mask[p] = 0;
        if (q + c.back() <= ans) return;
        cur[q++] = p;
        vector<int> nr, nc;
        bitset<N> nmask = mask & a[p];
        for (int i : r)
            if (a[p][i]) nr.push_back(i);
        if (!nr.empty()) {
            if (l < 4) {
                for (int i : nr)
                    d[i] = (a[i] & nmask).count();
                sort(nr.begin(), nr.end(),
                    [&](int x, int y) { return d[x] > d[y]; });
            }
            csort(nr, nc), dfs(nr, nc, l + 1, nmask);
        } else if (q > ans) ans = q, copy_n(cur, q, sol);
        c.pop_back(), q--;
    }
}
int solve(bitset<N> mask = bitset<N>(),
    string(N, '1')) { // vertex mask
    vector<int> r, c;
    ans = q = 0;
    for (int i = 0; i < n; i++)
        if (mask[i]) r.push_back(i);
    for (int i = 0; i < n; i++)
        d[i] = (a[i] & mask).count();
    sort(r.begin(), r.end(),
        [&](int i, int j) { return d[i] > d[j]; });
    csort(r, c), dfs(r, c, 1, mask);
    return ans; // sol[0 ~ ans-1]
}
} graph;

```

2.7 Minimum Steiner Tree*

```

// Minimum Steiner Tree
// O(V 3^T + V^2 2^T)
struct SteinerTree { // 0-base
    static const int T = 10, N = 105, INF = 1e9;
    int n, dst[N][N], dp[1 << T][N], tdst[N];
    int vcost[N]; // the cost of vertices
    void init(int _n) {
        n = _n;
        for (int i = 0; i < n; ++i) {
            for (int j = 0; j < n; ++j) dst[i][j] = INF;
            dst[i][i] = vcost[i] = 0;
        }
    }
    void add_edge(int ui, int vi, int wi) {
        dst[ui][vi] = min(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)
                dst[i][j] =
                    min(dst[i][j], dst[i][k] + dst[k][j]);
}
int solve(const vector<int> &ter) {
    shortest_path();
    int t = SZ(ter);
    for (int i = 0; i < (1 << t); ++i)
        for (int j = 0; j < n; ++j) dp[i][j] = INF;
    for (int i = 0; i < n; ++i) dp[0][i] = vcost[i];
    for (int msk = 1; msk < (1 << t); ++msk) {
        if (!(msk & (msk - 1))) {
            int who = __lg(msk);
            for (int i = 0; i < n; ++i)
                dp[msk][i] =
                    vcost[ter[who]] + dst[ter[who]][i];
        }
        for (int i = 0; i < n; ++i)
            for (int submsk = (msk - 1) & msk; submsk;
                submsk = (submsk - 1) & msk)
                dp[msk][i] = min(dp[msk][i],
                    dp[submsk][i] + dp[msk ^ submsk][i] -
                    vcost[i]);
        for (int i = 0; i < n; ++i) {
            tdst[i] = INF;
            for (int j = 0; j < n; ++j)
                tdst[i] =
                    min(tdst[i], dp[msk][j] + dst[j][i]);
        }
        for (int i = 0; i < n; ++i) dp[msk][i] = tdst[i];
    }
    int ans = INF;
    for (int i = 0; i < n; ++i)
        ans = min(ans, dp[(1 << t) - 1][i]);
    return ans;
}
};

```

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*

```

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

```

2.10 Vizing's theorem

```

namespace vizing { // returns edge coloring in adjacent
// matrix G. 1 - based
int C[kN][kN], G[kN][kN];
void clear(int N) {
    for (int i = 0; i <= N; i++) {
        for (int j = 0; j <= N; j++) C[i][j] = G[i][j] = 0;
    }
}

```

```

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

```

2.11 Minimum Clique Cover*

```

struct Clique_Cover { // 0-base,  $O(n^2 \cdot n)$ 
    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);
for (int ans = 1; ans < n; ++ans) {
    int sum = 0;
    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;
    }
};

```

2.13 Dijkstra

```

vector<pii> edge[mxn];
int dis[mxn], vis[mxn];
void dijkstra(int s)
{
    memset(dis, 0x7f, sizeof(dis));
    dis[s] = 0;
    priority_queue<pii, vector<pii>, greater<pii>> pq;
    pq.emplace(0, s);
    while (pq.size()) {
        int now = pq.top().Y;
        pq.pop();
        if (vis[now]) continue;
        vis[now] = 1;
        for (pii e : edge[now]) {
            if (!vis[e.X] && dis[e.X] > dis[now] + e.Y) {
                dis[e.X] = dis[now] + e.Y;
                pq.emplace(dis[e.X], e.X);
            }
        }
    }
}

```

2.14 Theory

Maximum independent edge set = $|V| - \text{Minimum edge cover}$
 Maximum independent set = $|V| - \text{Minimum vertex cover}$
 A sequence of non-negative 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 k in $1 \leq k \leq n$.

3 Data Structure

3.1 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;
}
```

3.2 Heavy light Decomposition

```
struct Heavy_light_Decomposition { // 1-base
    int n, ulink[10005], deep[10005], mxson[10005],
        w[10005], pa[10005];
    int t, pl[10005], data[10005], dt[10005], bln[10005],
        edge[10005], et;
    vector<pii> G[10005];
    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;
}
```

3.3 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;
    }
};
```


3.4 Treap

```

struct node {
    int data, sz;
    node *l, *r;
    node(int k) : data(k), sz(1), l(0), r(0) {}
    void up() {
        sz = 1;
        if (l) sz += l->sz;
        if (r) sz += r->sz;
    }
    void down() {}
};

int sz(node *a) { return a ? a->sz : 0; }
node *merge(node *a, node *b) {
    if (!a || !b) return a ? a : b;
    if (rand() % (sz(a) + sz(b)) < sz(a))
        return a->down(), a->r = merge(a->r, b), a->up(),
            a;
    return b->down(), b->l = merge(a, b->l), b->up(), b;
}

void split(node *o, node *&a, node *&b, int k) {
    if (!o) return a = b = 0, void();
    o->down();
    if (o->data <= k)
        a = o, split(o->r, a->r, b, k), a->up();
    else b = o, split(o->l, a, b->l, k), b->up();
}

void split2(node *o, node *&a, node *&b, int k) {
    if (sz(o) <= k) return a = o, b = 0, void();
    o->down();
    if (sz(o->l) + 1 <= k)
        a = o, split2(o->r, a->r, b, k - sz(o->l) - 1);
    else b = o, split2(o->l, a, b->l, k);
    o->up();
}

node *kth(node *o, int k) {
    if (k <= sz(o->l)) return kth(o->l, k);
    if (k == sz(o->l) + 1) return o;
    return kth(o->r, k - sz(o->l) - 1);
}

int Rank(node *o, int key) {
    if (o->data < key)
        return sz(o->l) + 1 + Rank(o->r, key);
    else return Rank(o->l, key);
}

bool erase(node *&o, int k) {
    if (!o) return 0;
    if (o->data == k) {
        node *t = o;
        o->down(), o = merge(o->l, o->r);
        delete t;
        return 1;
    }
    node *&t = k < o->data ? o->l : o->r;
    return erase(t, k) ? o->up(), 1 : 0;
}

void insert(node *&o, int k) {
    node *a, *b;
    split(o, a, b, k),
        o = merge(a, merge(new node(k), b));
}

void interval(node *&o, int l, int r) {
    node *a, *b, *c;
    split2(o, a, b, l - 1), split2(b, b, c, r);
    // operate
    o = merge(a, merge(b, c));
}

```

3.5 RangeOperations

```

struct node {
    ll sum;
    ll mx, mxcnt, smx;
    ll mi, micnt, smi;
    ll lazymax, lazymin, lazyadd;
    node(ll k = 0)
        : sum(k), mx(k), mxcnt(1), smx(-INF), mi(k),
          micnt(1), smi(INF), lazymax(-INF), lazymin(INF),

```

```

    lazyadd(0) {}
    node operator+(const node &a) const {
        node rt;
        rt.sum = sum + a.sum;
        rt.mx = max(mx, a.mx);
        rt.mi = min(mi, a.mi);
        if (mx == a.mx) {
            rt.mxcnt = mxcnt + a.mxcnt;
            rt.smx = max(smx, a.smx);
        } else if (mx > a.mx) {
            rt.mxcnt = mxcnt;
            rt.smx = max(smx, a.mx);
        } else {
            rt.mxcnt = a.mxcnt;
            rt.smx = max(mx, a.smx);
        }
        if (mi == a.mi) {
            rt.micnt = micnt + a.micnt;
            rt.smi = min(smi, a.smi);
        } else if (mi < a.mi) {
            rt.micnt = micnt;
            rt.smi = min(smi, a.mi);
        } else {
            rt.micnt = a.micnt;
            rt.smi = min(mi, a.smi);
        }
        rt.lazymax = -INF;
        rt.lazymin = INF;
        rt.lazyadd = 0;
        return rt;
    }
} seg[MAXC << 2];
ll a[MAXC];

void give_tag_min(int rt, ll t) {
    if (t >= seg[rt].mx) return;
    seg[rt].lazymin = t;
    seg[rt].lazymax = min(seg[rt].lazymax, t);
    seg[rt].sum -= seg[rt].mxcnt * (seg[rt].mx - t);
    if (seg[rt].mx == seg[rt].smi) seg[rt].smi = t;
    if (seg[rt].mx == seg[rt].mi) seg[rt].mi = t;
    seg[rt].mx = t;
}

void give_tag_max(int rt, ll t) {
    if (t <= seg[rt].mi) return;
    seg[rt].lazymax = t;
    seg[rt].sum += seg[rt].micnt * (t - seg[rt].mi);
    if (seg[rt].mi == seg[rt].smx) seg[rt].smx = t;
    if (seg[rt].mi == seg[rt].mx) seg[rt].mx = t;
    seg[rt].mi = t;
}

void give_tag_add(int l, int r, int rt, ll t) {
    seg[rt].lazyadd += t;
    if (seg[rt].lazymax != -INF) seg[rt].lazymax += t;
    if (seg[rt].lazymin != INF) seg[rt].lazymin += t;
    seg[rt].mx += t;
    if (seg[rt].smx != -INF) seg[rt].smx += t;
    seg[rt].mi += t;
    if (seg[rt].smi != INF) seg[rt].smi += t;
    seg[rt].sum += (ll)(r - l + 1) * t;
}

void tag_down(int l, int r, int rt) {
    if (seg[rt].lazyadd != 0) {
        int mid = (l + r) >> 1;
        give_tag_add(l, mid, rt << 1, seg[rt].lazyadd);
        give_tag_add(
            mid + 1, r, rt << 1 | 1, seg[rt].lazyadd);
        seg[rt].lazyadd = 0;
    }
    if (seg[rt].lazymin != INF) {
        give_tag_min(rt << 1, seg[rt].lazymin);
        give_tag_min(rt << 1 | 1, seg[rt].lazymin);
        seg[rt].lazymin = INF;
    }
    if (seg[rt].lazymax != -INF) {
        give_tag_max(rt << 1, seg[rt].lazymax);
        give_tag_max(rt << 1 | 1, seg[rt].lazymax);
        seg[rt].lazymax = -INF;
    }
}

void build(int l, int r, int rt) {
    if (l == r) return seg[rt] = node(a[l]), void();
    int mid = (l + r) >> 1;

```

```

    build(l, mid, rt << 1);
    build(mid + 1, r, rt << 1 | 1);
    seg[rt] = seg[rt << 1] + seg[rt << 1 | 1];
}
void modifymax(
    int L, int R, int l, int r, int rt, ll t) {
    if (L <= l && R >= r && t < seg[rt].smi)
        return give_tag_max(rt, t);
    if (l != r) tag_down(l, r, rt);
    int mid = (l + r) >> 1;
    if (L <= mid) modifymax(L, R, l, mid, rt << 1, t);
    if (R > mid)
        modifymax(L, R, mid + 1, r, rt << 1 | 1, t);
    seg[rt] = seg[rt << 1] + seg[rt << 1 | 1];
}
void modifymin(
    int L, int R, int l, int r, int rt, ll t) {
    if (L <= l && R >= r && t > seg[rt].smx)
        return give_tag_min(rt, t);
    if (l != r) tag_down(l, r, rt);
    int mid = (l + r) >> 1;
    if (L <= mid) modifymin(L, R, l, mid, rt << 1, t);
    if (R > mid)
        modifymin(L, R, mid + 1, r, rt << 1 | 1, t);
    seg[rt] = seg[rt << 1] + seg[rt << 1 | 1];
}
void modifyadd(
    int L, int R, int l, int r, int rt, ll t) {
    if (L <= l && R >= r)
        return give_tag_add(l, r, rt, t);
    if (l != r) tag_down(l, r, rt);
    int mid = (l + r) >> 1;
    if (L <= mid) modifyadd(L, R, l, mid, rt << 1, t);
    if (R > mid)
        modifyadd(L, R, mid + 1, r, rt << 1 | 1, t);
    seg[rt] = seg[rt << 1] + seg[rt << 1 | 1];
}
ll query(int L, int R, int l, int r, int rt) {
    if (L <= l && R >= r) return seg[rt].sum;
    if (l != r) tag_down(l, r, rt);
    int mid = (l + r) >> 1;
    if (R <= mid) return query(L, R, l, mid, rt << 1);
    if (L > mid)
        return query(L, R, mid + 1, r, rt << 1 | 1);
    return query(L, R, l, mid, rt << 1) +
        query(L, R, mid + 1, r, rt << 1 | 1);
}
int main() {
    int n, m;
    cin >> n >> m;
    for (int i = 1; i <= n; ++i) cin >> a[i];
    build(1, n, 1);
    while (m--) {
        int k, x, y;
        ll t;
        cin >> k >> x >> y, ++x;
        if (k == 0) cin >> t, modifymin(x, y, 1, n, 1, t);
        else if (k == 1)
            cin >> t, modifymax(x, y, 1, n, 1, t);
        else if (k == 2)
            cin >> t, modifyadd(x, y, 1, n, 1, t);
        else cout << query(x, y, 1, n, 1) << "\n";
    }
}

```

4 Flow/Matching

4.1 Kuhn Munkres

```

struct KM { // 0-base
    int w[MAXN][MAXN], hl[MAXN], hr[MAXN], slk[MAXN], n;
    int fl[MAXN], fr[MAXN], pre[MAXN], qu[MAXN], ql, qr;
    bool vl[MAXN], vr[MAXN];
    void init(int _n) {
        n = _n;
        for (int i = 0; i < n; ++i)
            for (int j = 0; j < n; ++j) w[i][j] = -INF;
    }
}

```

```

void add_edge(int a, int b, int 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(slk, slk + n, INF);
    fill(vl, vl + n, 0), fill(vr, vr + n, 0);
    ql = qr = 0, qu[qr++] = s, vr[s] = 1;
    while (1) {
        int 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;
    }
}
int Solve() {
    fill(fl, fl + n, -1), fill(fr, fr + n, -1),
    fill(hr, hr + n, 0);
    for (int i = 0; i < n; ++i)
        hl[i] = *max_element(w[i], w[i] + n);
    for (int i = 0; i < n; ++i) Bfs(i);
    int res = 0;
    for (int i = 0; i < n; ++i) res += w[i][fl[i]];
    return res;
}
};

```

4.2 MincostMaxflow

```

struct MCMF { // 0-base
    struct edge {
        ll from, to, cap, flow, cost, rev;
    } * past[MAXN];
    vector<edge> G[MAXN];
    bitset<MAXN> inq;
    ll dis[MAXN], up[MAXN], s, t, mx, n;
    bool BellmanFord(ll &flow, ll &cost) {
        fill(dis, dis + n, INF);
        queue<ll> q;
        q.push(s), inq.reset(), inq[s] = 1;
        up[s] = mx - flow, past[s] = 0, dis[s] = 0;
        while (!q.empty()) {
            ll u = q.front();
            q.pop(), inq[u] = 0;
            if (!up[u]) continue;
            for (auto &e : G[u])
                if (e.flow != e.cap &&
                    dis[e.to] > dis[u] + e.cost) {
                    dis[e.to] = dis[u] + e.cost, past[e.to] = &e;
                    up[e.to] = min(up[u], e.cap - e.flow);
                    if (!inq[e.to]) inq[e.to] = 1, q.push(e.to);
                }
        }
        if (dis[t] == INF) return 0;
        flow += up[t], cost += up[t] * dis[t];
        for (ll i = t; past[i]; i = past[i]->from) {
            auto &e = *past[i];
            e.flow += up[t], G[e.to][e.rev].flow -= up[t];
        }
        return 1;
    }
}
ll MinCostMaxFlow(ll _s, ll _t, ll &cost) {
}

```



```

    s = _s, t = _t, cost = 0;
    ll flow = 0;
    while (BellmanFord(flow, cost))
        ;
    return flow;
}
void init(ll _n, ll _mx) {
    n = _n, mx = _mx;
    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, G[b].size()});
    G[b].pb(edge{b, a, 0, 0, -cost, G[a].size() - 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

```

// global min cut
struct SW { // O(V^3)
    static const int MXN = 514;
    int n, vst[MXN], del[MXN];
    int edge[MXN][MXN], wei[MXN];
    void init(int _n) {
        n = _n, MEM(edge, 0), MEM(del, 0);
    }
};

```

```

void addEdge(int u, int v, int w) {
    edge[u][v] += w, edge[v][u] += w;
}
void search(int &s, int &t) {
    MEM(vst, 0), MEM(wei, 0), s = t = -1;
    while (1) {
        int mx = -1, cur = 0;
        for (int i = 0; i < n; ++i)
            if (!del[i] && !vst[i] && mx < wei[i])
                cur = i, mx = wei[i];
        if (mx == -1) break;
        vst[cur] = 1, s = t, t = cur;
        for (int i = 0; i < n; ++i)
            if (!vst[i] && !del[i]) wei[i] += edge[cur][i];
    }
}
int solve() {
    int res = INF;
    for (int i = 0, x, y; i < n - 1; ++i) {
        search(x, y), res = min(res, wei[y]), del[y] = 1;
        for (int j = 0; j < n; ++j)
            edge[x][j] = (edge[j][x] += edge[y][j]);
    }
    return res;
}
};

```

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

```

struct Gomory_Hu_tree { // 0-base
    MaxFlow Dinic;
    int n;
    vector<pii> G[MAXN];
    void init(int _n) {
        n = _n;
        for (int i = 0; i < n; ++i) G[i].clear();
    }
    void solve(vector<int> &v) {
        if (v.size() <= 1) return;
        int s = rand() % SZ(v);
        swap(v.back(), v[s]), s = v.back();
        int t = v[rand() % (SZ(v) - 1)];
        vector<int> L, R;
        int x = (Dinic.reset(), Dinic.maxflow(s, t));
        G[s].pb(pii(t, x)), G[t].pb(pii(s, x));
        for (int i : v)
            if (~Dinic.dis[i]) L.pb(i);
            else R.pb(i);
        solve(L), solve(R);
    }
    void build() {
        vector<int> v(n);
        for (int i = 0; i < n; ++i) v[i] = i;
        solve(v);
    }
} ght; // test by BZOJ 4519
MaxFlow &Dinic = ght.Dinic;

```

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

```
const int MAXN = 1e5 + 5;
int z[MAXN];
void make_z(string s) {
    int l = 0, r = 0;
    for (int i = 1; i < s.size(); i++) {
        for (z[i] = max(0, min(r - i + 1, z[i - 1]));
             i + z[i] < s.size() && 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];
int Manacher(string tmp) {
    string s = "&";
    int l = 0, r = 0, x, ans;
    for (char c : tmp) s.pb(c), s.pb('%');
    ans = 0, x = 0;
    for (int i = 1; i < SZ(s); ++i) {
        z[i] = r > i ? min(z[2 * l - i], r - i) : 1;
        while (s[i + z[i]] == s[i - z[i]]) ++z[i];
        if (z[i] + i > r) r = z[i] + i, l = i;
    }
    for (int i = 1; i < SZ(s); ++i)
        if (s[i] == '%') x = max(x, z[i]);
    ans = x / 2 * 2, x = 0;
    for (int i = 1; i < SZ(s); ++i)
        if (s[i] != '%') x = max(x, z[i]);
    return max(ans, (x - 1) / 2 * 2 + 1);
}
```

5.4 Suffix Array

```
struct suffix_array {
    int box[MAXN], tp[MAXN], m;
    bool not_equ(int a, int b, int k, int n) {
        return ra[a] != ra[b] || a + k >= n ||
            b + k >= n || ra[a + k] != ra[b + k];
    }
    void radix(int *key, int *it, int *ot, int n) {
        fill_n(box, m, 0);
        for (int i = 0; i < n; ++i) ++box[key[i]];
        partial_sum(box, box + m, box);
        for (int i = n - 1; i >= 0; --i)
            ot[--box[key[it[i]]]] = it[i];
    }
    void make_sa(string s, int n) {
        int k = 1;
        for (int i = 0; i < n; ++i) ra[i] = s[i];
        do {
            iota(tp, tp + k, n - k), iota(sa + k, sa + n, 0);
            radix(ra + k, sa + k, tp + k, n - k);
            radix(ra, tp, sa, n);
            tp[sa[0]] = 0, m = 1;
            for (int i = 1; i < n; ++i) {
                m += not_equ(sa[i], sa[i - 1], k, n);
                tp[sa[i]] = m - 1;
            }
            copy_n(tp, n, ra);
            k *= 2;
        } while (k < n && m != n);
    }
    void make_he(string s, int n) {
        for (int j = 0, k = 0; j < n; ++j) {
            if (ra[j])
                for (; s[j + k] == s[sa[ra[j]] - 1] + k; ++k)
                    ;
            he[ra[j]] = k, k = max(0, k - 1);
        }
    }
    void make_fa(int n) {
        for (int i = 1; i <= n; ++i) fa[0][i] = he[i - 1];
    }
}
```

```
for (int j = 1; j < 20; ++j)
    for (int i = 1; i + (1 < j) - 1 <= n; ++i)
        fa[j][i] = min(fa[j - 1][i], fa[j - 1][i + (1 < (j - 1))]);
}
inline int ask(int x, int y) {
    int k = Log[y - x + 1];
    return min(fa[k][x], fa[k][y - (1 < k) + 1]);
}
inline int lcp(int x, int y) {
    if (x > y) swap(x, y);
    return ask(x + 2, y + 1);
}
int sa[MAXN], ra[MAXN], he[MAXN], fa[20][MAXN], Log[
    MAXN];
void build(const string &s) {
    int n = sz(s);
    fill_n(sa, n, 0), fill_n(ra, n, 0), fill_n(he, n,
        0);
    fill_n(box, n, 0), fill_n(tp, n, 0), m = 256;
    Log[0] = Log[1] = 0;
    for (int i = 2; i <= n; ++i) Log[i] = Log[i >> 1] + 1;
    make_sa(s, sz(s));
    make_he(s, sz(s));
    make_fa(sz(s));
}
};
```

5.5 SAIS*

```
class SAIS {
public:
    int *SA, *H;
    // zero based, string content MUST > 0
    // result height H[i] is LCP(SA[i - 1], SA[i])
    // string, length, |sigma|
    void build(int *s, int n, int m = 128) {
        copy_n(s, n, _s);
        _h[0] = _s[n++] = 0;
        sais(_s, _sa, _p, _q, _t, _c, n, m);
        mkhei(n);
        SA = _sa + 1;
        H = _h + 1;
    }
private:
    bool _t[N * 2];
    int _s[N * 2], _c[N * 2], x[N], _p[N], _q[N * 2],
        r[N], _sa[N * 2], _h[N];
    void mkhei(int n) {
        for (int i = 0; i < n; i++) r[_sa[i]] = i;
        for (int i = 0; i < n; i++)
            if (r[i]) {
                int ans = i > 0 ? max(_h[r[i] - 1] - 1, 0) : 0;
                while (_s[i + ans] == _s[_sa[r[i] - 1] + ans])
                    ans++;
                _h[r[i]] = ans;
            }
    }
    void sais(int *s, int *sa, int *p, int *q, bool *t,
        int *c, int n, int z) {
        bool uniq = t[n - 1] = 1, neq;
        int nn = 0, nmxx = -1, *nsa = sa + n, *ns = s + n,
            lst = -1;

#define MAGIC(XD) \
        fill_n(sa, n, 0); \
        copy_n(c, z, x); \
        XD; \
        copy_n(c, z - 1, x + 1); \
        for (int i = 0; i < n; i++) \
            if (sa[i] && !t[sa[i] - 1]) \
                sa[x[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[sa[i] - 1]] = sa[i] - 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]);
MAGIC(for (int i = 1; i <= n - 1;
          i++) if (t[i] && !t[i - 1])
        sa[--x[s[i]]] = p[q[i] = nn++] = i);
for (int i = 0; i < n; i++)
    if (sa[i] && t[sa[i]] && !t[sa[i] - 1]) {
        neq = (lst < 0) ||
            !equal(s + lst,
                  s + lst + p[q[sa[i]] + 1] - sa[i],
                  s + sa[i]);
        ns[q[lst = sa[i]]] = nmzx += neq;
    }
sais(ns, nsa, p + nn, q + n, t + n, c + z, nn,
     nmzx + 1);
MAGIC(for (int i = nn - 1; i >= 0; i--)
        sa[--x[s[p[nsa[i]]]]] = p[nsa[i]]);
}
} sa;

```

5.6 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]];
    }
};

```

5.7 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);
}

```

5.8 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);
    }
} dbs;

```

5.9 SAM

```

const int MAXM = 1000010;
struct SAM {
    int tot, root, lst, mom[MAXM], mx[MAXM];
    int acc[MAXM], nxt[MAXM][33];
    int newNode() {
        int res = ++tot;
        fill(nxt[res], nxt[res] + 33, 0);
        mom[res] = mx[res] = acc[res] = 0;
        return res;
    }
    void init() {
        tot = 0;
        root = newNode();
        mom[root] = 0, mx[root] = 0;
        lst = root;
    }
    void push(int c) {
        int p = lst;
        int np = newNode();
        mx[np] = mx[p] + 1;
        for (; p && nxt[p][c] == 0; p = mom[p])
            nxt[p][c] = np;
        if (p == 0) mom[np] = root;
        else {
            int q = nxt[p][c];
            if (mx[p] + 1 == mx[q]) mom[np] = q;
            else {
                int nq = newNode();
                mx[nq] = mx[p] + 1;
                for (int i = 0; i < 33; i++)
                    nxt[nq][i] = nxt[q][i];
                mom[nq] = mom[q];
                mom[q] = nq;
                mom[np] = nq;
                for (; p && nxt[p][c] == q; p = mom[p])
                    nxt[p][c] = nq;
            }
        }
        lst = np;
    }
    void push(char *str) {
        for (int i = 0; str[i]; i++)

```

```

    push(str[i] - 'a' + 1);
}
} sam;

```

```

// recover a
a[al] = '\0';
return clcs;
}

```

5.10 cyclicLCS

```

#define L 0
#define LU 1
#define U 2
const int mov[3][2] = {0, -1, -1, -1, -1, 0};
int al, bl;
char a[MAXL * 2], b[MAXL * 2]; // 0-indexed
int dp[MAXL * 2][MAXL];
char pred[MAXL * 2][MAXL];
inline int lcs_length(int r) {
    int i = r + al, j = bl, l = 0;
    while (i > r) {
        char dir = pred[i][j];
        if (dir == LU) l++;
        i += mov[dir][0];
        j += mov[dir][1];
    }
    return l;
}
inline void reroot(int r) { // r = new base row
    int i = r, j = 1;
    while (j <= bl && pred[i][j] != LU) j++;
    if (j > bl) return;
    pred[i][j] = L;
    while (i < 2 * al && j <= bl) {
        if (pred[i + 1][j] == U) {
            i++;
            pred[i][j] = L;
        } else if (j < bl && pred[i + 1][j + 1] == LU) {
            i++;
            j++;
            pred[i][j] = L;
        } else {
            j++;
        }
    }
}
int cyclic_lcs() {
    // a, b, al, bl should be properly filled
    // note: a WILL be altered in process
    // -- concatenated after itself
    char tmp[MAXL];
    if (al > bl) {
        swap(al, bl);
        strcpy(tmp, a);
        strcpy(a, b);
        strcpy(b, tmp);
    }
    strcpy(tmp, a);
    strcat(a, tmp);
    // basic lcs
    for (int i = 0; i <= 2 * al; i++) {
        dp[i][0] = 0;
        pred[i][0] = U;
    }
    for (int j = 0; j <= bl; j++) {
        dp[0][j] = 0;
        pred[0][j] = L;
    }
    for (int i = 1; i <= 2 * al; i++) {
        for (int j = 1; j <= bl; j++) {
            if (a[i - 1] == b[j - 1])
                dp[i][j] = dp[i - 1][j - 1] + 1;
            else dp[i][j] = max(dp[i - 1][j], dp[i][j - 1]);
            if (dp[i][j - 1] == dp[i][j]) pred[i][j] = L;
            else if (a[i - 1] == b[j - 1]) pred[i][j] = LU;
            else pred[i][j] = U;
        }
    }
    // do cyclic lcs
    int clcs = 0;
    for (int i = 0; i < al; i++) {
        clcs = max(clcs, lcs_length(i));
        reroot(i + 1);
    }
}

```

6 Math

6.1 ax+by=gcd*

```

pll exgcd(ll a, ll b) {
    if (b == 0) return pll(1, 0);
    else {
        ll p = a / b;
        pll q = exgcd(b, a % b);
        return pll(q.Y, q.X - q.Y * p);
    }
}

```

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 sum*

```

ll floor_sum(ll n, ll m, ll a, ll b) {
    ll ans = 0;
    if (a >= m)
        ans += (n - 1) * n * (a / m) / 2, a %= m;
    if (b >= m)
        ans += n * (b / m), b %= m;
    ll y_max = (a * n + b) / m, x_max = (y_max * m - b) / a;
    if (y_max == 0) return ans;
    ans += (n - (x_max + a - 1) / a) * y_max;
    ans += floor_sum(y_max, a, m, (a - x_max % a) % a);
    return ans;
} // sum^{n-1}_0 floor((a * i + b) / m) in log(n + m + a + b)

```

6.4 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 : pirms <= 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 & 1) ^ 1) 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;
}

```

6.5 Big number

```

template<typename T>
inline string to_string(const T& x) {
    stringstream ss;
    return ss << x, ss.str();
}

```

```

}
struct bigN:vector<ll>{
    const static int base=1000000000,width=log10(base);
    bool negative;
    bigN(const_iterator a,const_iterator b):vector<ll>(a,
        b){}
    bigN(string s){
        if(s.empty())return;
        if(s[0]=='-')negative=1,s=s.substr(1);
        else negative=0;
        for(int i=int(s.size())-1;i>=0;i-=width){
            ll t=0;
            for(int j=max(0,i-width+1);j<=i;++j)
                t=t*10+s[j]-'0';
            push_back(t);
        }
        trim();
    }
    template<typename T>
    bigN(const T &x):bigN(to_string(x)){}
    bigN():negative(0){}
    void trim(){
        while(size()&&!back())pop_back();
        if(empty())negative=0;
    }
    void carry(int _base=base){
        for(size_t i=0;i<size();++i){
            if(at(i)>=0&&at(i)<_base)continue;
            if(i+1u==size())push_back(0);
            int r=at(i)%_base;
            if(r<0)r+=_base;
            at(i+1)+=(at(i)-r)/_base,at(i)=r;
        }
    }
    int abscmp(const bigN &b)const{
        if(size()>b.size())return 1;
        if(size()<b.size())return -1;
        for(int i=int(size())-1;i>=0;--i){
            if(at(i)>b[i])return 1;
            if(at(i)<b[i])return -1;
        }
        return 0;
    }
    int cmp(const bigN &b)const{
        if(negative!=b.negative)return negative?-1:1;
        return negative?-abscmp(b):abscmp(b);
    }
    bool operator<(const bigN&b)const{return cmp(b)<0;}
    bool operator>(const bigN&b)const{return cmp(b)>0;}
    bool operator<=(const bigN&b)const{return cmp(b)<=0;}
    bool operator>=(const bigN&b)const{return cmp(b)>=0;}
    bool operator==(const bigN&b)const{return !cmp(b);}
    bool operator!=(const bigN&b)const{return cmp(b)!=0;}
    bigN abs()const{
        bigN res=*this;
        return res.negative=0, res;
    }
    bigN operator-()const{
        bigN res=*this;
        return res.negative=!negative,res.trim(),res;
    }
    bigN operator+(const bigN &b)const{
        if(negative)return -(*this)+(-b);
        if(b.negative)return *this-(-b);
        bigN res=*this;
        if(b.size()>size())res.resize(b.size());
        for(size_t i=0;i<b.size();++i)res[i]+=b[i];
        return res.carry(),res.trim(),res;
    }
    bigN operator-(const bigN &b)const{
        if(negative)return -(*this)-(-b);
        if(b.negative)return *this+(-b);
        if(abscmp(b)<0)return -(b-(*this));
        bigN res=*this;
        if(b.size()>size())res.resize(b.size());
        for(size_t i=0;i<b.size();++i)res[i]-=b[i];
        return res.carry(),res.trim(),res;
    }
    bigN operator*(const bigN &b)const{
        bigN res;
        res.negative=negative!=b.negative;
        res.resize(size()+b.size());

```

```

        for(size_t i=0;i<size();++i)
            for(size_t j=0;j<b.size();++j)
                if((res[i+j]+=at(i)*b[j])>=base){
                    res[i+j+1]+=res[i+j]/base;
                    res[i+j]%=base;
                }
            return res.trim(),res;
        }
        bigN operator/(const bigN &b)const{
            int norm=base/(b.back()+1);
            bigN x=abs()*norm;
            bigN y=b.abs()*norm;
            bigN q,r;
            q.resize(x.size());
            for(int i=int(x.size())-1;i>=0;--i){
                r=r*base+x[i];
                int s1=r.size()<=y.size()?0:r[y.size()];
                int s2=r.size()<y.size()?0:r[y.size()-1];
                int d=(ll(base)*s1+s2)/y.back();
                r=r-y*d;
                while(r.negative)r=r+y,--d;
                q[i]=d;
            }
            q.negative=negative!=b.negative;
            return q.trim(),q;
        }
        bigN operator%(const bigN &b)const{
            return *this-(*this/b)*b;
        }
        friend istream& operator>>(istream &ss, bigN &b){
            string s;
            return ss>>s, b=s, ss;
        }
        friend ostream& operator<<(ostream &ss, const bigN &b)
        {
            if(b.negative)ss<<"-";
            ss<<(b.empty()?0:b.back());
            for(int i=int(b.size())-2;i>=0;--i)
                ss<<setw(width)<<setfill('0')<<b[i];
            return ss;
        }
        template<typename T>
        operator T(){
            stringstream ss;
            ss<<*this;
            T res;
            return ss>>res,res;
        }
    };

```

6.6 Fraction

```

struct fraction{
    ll n,d;
    fraction(const ll &n=0,const ll &d=1):n(_n),d(_d){
        ll t=__gcd(n,d);
        n/=t,d/=t;
        if(d<0) n=-n,d=-d;
    }
    fraction operator-()const{
        return fraction(-n,d);
    }
    fraction operator+(const fraction &b)const{
        return fraction(n*b.d+b.n*d,d*b.d);
    }
    fraction operator-(const fraction &b)const{
        return fraction(n*b.d-b.n*d,d*b.d);
    }
    fraction operator*(const fraction &b)const{
        return fraction(n*b.n,d*b.d);
    }
    fraction operator/(const fraction &b)const{
        return fraction(n*b.d,d*b.n);
    }
    void print(){
        cout << n;
        if(d!=1) cout << "/" << d;
    }
};

```


6.7 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

```
// does not work when n is prime
ll f(ll x, ll mod){ return add(mul(x,x,mod),1,mod); }
ll pollard_rho(ll n){
    if(!(n&1)) return 2;
    while(1){
        ll y=2,x=rand()%(n-1)+1,res=1;
        for(int sz=2;res==1;y=x,sz*=2)
            for(int i=0;i<sz&&res<=1;++i)
                x=f(x,n),res=__gcd(abs(x-y),n);
        if(res!=0&&res!=n) return res;
    }
}
```

6.9 Simplex Algorithm

```
const int MAXN = 111;
const int MAXM = 111;
const double eps = 1E-10;
double a[MAXN][MAXM], b[MAXN], c[MAXM], d[MAXN][MAXM];
double 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(double a[MAXN][MAXM], double b[MAXN],
    double c[MAXM], int n, int m){
    ++m;
    int r = n, s = m - 1;
    memset(d, 0, sizeof(d));
    for (int i = 0; i < n + m; ++i) ix[i] = i;
    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;
    }
    for (int j = 0; j < m - 1; ++j) d[n][j] = c[j];
    d[n + 1][m - 1] = -1;
    for (double dd;; ) {
        if (r < n) {
            int t = ix[s]; ix[s] = ix[r + m]; ix[r + m] = t;
            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 = -1; 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;
for(int i=0; i<m; i++) x[i] = 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;
}
```

6.10 Schreier-Sims Algorithm*

```
namespace schreier {
    int n;
    vector<vector<vector<int>>> bkets, binv;
    vector<vector<int>> lk;
    vector<int> operator*(const vector<int> &a, const
        vector<int> &b) {
        vector<int> res(SZ(a));
        for (int i = 0; i < SZ(a); ++i) res[i] = b[a[i]];
        return res;
    }
    vector<int> inv(const vector<int> &a) {
        vector<int> res(SZ(a));
        for (int i = 0; i < SZ(a); ++i) res[a[i]] = i;
        return res;
    }
    int filter(const vector<int> &g, bool add = true) {
        n = SZ(bkets);
        vector<int> p = g;
        for (int i = 0; i < n; ++i) {
            assert(p[i] >= 0 && p[i] < SZ(lk[i]));
            if (lk[i][p[i]] == -1) {
                if (add) {
                    bkets[i].pb(p);
                    binv[i].pb(inv(p));
                    lk[i][p[i]] = SZ(bkets[i]) - 1;
                }
                return i;
            }
            p = p * binv[i][lk[i][p[i]]];
        }
        return -1;
    }
    bool inside(const vector<int> &g) { return filter(g,
        false) == -1; }
    void solve(const vector<vector<int>> &gen, int _n) {
        n = _n;
        bkets.clear(), bkets.resize(n);
        binv.clear(), binv.resize(n);
        lk.clear(), lk.resize(n);
        vector<int> iden(n);
        iota(iden.begin(), iden.end(), 0);
        for (int i = 0; i < n; ++i) {
```

```

        lk[i].resize(n, -1);
        bkts[i].pb(iden);
        binv[i].pb(iden);
        lk[i][i] = 0;
    }
    for (int i = 0; i < SZ(gen); ++i) filter(gen[i]);
    queue<pair<pii, pii>> upd;
    for (int i = 0; i < n; ++i)
        for (int j = i; j < n; ++j)
            for (int k = 0; k < SZ(bkts[i]); ++k)
                for (int l = 0; l < SZ(bkts[j]); ++l)
                    upd.emplace(pii(i, k), pii(j, l));
    while (!upd.empty()) {
        auto a = upd.front().X;
        auto b = upd.front().Y;
        upd.pop();
        int res = filter(bkts[a.X][a.Y] * bkts[b.X][b.Y]);
        if (res == -1) continue;
        pii pr = pii(res, SZ(bkts[res]) - 1);
        for (int i = 0; i < n; ++i)
            for (int j = 0; j < SZ(bkts[i]); ++j) {
                if (i <= res) upd.emplace(pii(i, j), pr);
                if (res <= i) upd.emplace(pr, pii(i, j));
            }
    }
}
long long size() {
    long long res = 1;
    for (int i = 0; i < n; ++i) res = res * SZ(bkts[i]);
    return res;
}
}

```

6.11 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;
    pair<LL, LL> p = gcd(m1, m2);
    LL lcm = m1 * m2 * g;
    LL res = p.first * (x2 - x1) * m1 + x1;
    return (res % lcm + lcm) % lcm;
}

```

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 >>= 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

```

int64_t PrimeCount(int64_t n) {
    if (n <= 1) return 0;
    const int v = sqrt(n);
    vector<int> smalls(v + 1);
    for (int i = 2; i <= v; ++i) smalls[i] = (i + 1) / 2;
    int s = (v + 1) / 2;
    vector<int> roughs(s);
    for (int i = 0; i < s; ++i) roughs[i] = 2 * i + 1;
    vector<int64_t> larges(s);
    for (int i = 0; i < s; ++i) larges[i] = (n / (2 * i + 1) + 1) / 2;
    vector<bool> skip(v + 1);
    int pc = 0;
    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] = true;
            for (int i = q; i <= v; i += 2 * p) skip[i] = true;
            int ns = 0;
            for (int k = 0; k < s; ++k) {
                int i = roughs[k];
                if (skip[i]) continue;
                int64_t 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;
                for (int i = j * p, e = min(i + p, v + 1); i < e; ++i) smalls[i] -= c;
            }
        }
    }
    for (int k = 1; k < s; ++k) {
        const int64_t m = n / roughs[k];
        int64_t s = larges[k] - (pc + k - 1);
        for (int l = 1; l < k; ++l) {
            int p = roughs[l];
            if (1LL * p * p > m) break;
            s -= smalls[m / p] - (pc + l - 1);
        }
        larges[0] -= s;
    }
    return larges[0];
}

```

6.14 Primes

```

/*
12721 13331 14341 75577 123457 222557 556679 999983
1097774749 1076767633 100102021 999997771 1001010013
1000512343 987654361 999991231 999887733 98789101
987777733 999991921 1010101333 1010102101
100000000039 10000000000037 2305843009213693951
4611686018427387847 9223372036854775783
18446744073709551557
*/
bool sieve[mxn];

```

```
vector<int> prime;
void linear_sieve(){
    for(int i=2;i<mxn;++i){
        if(!sieve[i]) prime.pb(i);
        for(int p:prime){
            if(i*p>=mxn) break;
            sieve[i*p]=1;
            if(i%p==0) break;
        }
    }
}
```

6.15 PrimeFactorization

```
vector<pll> f(ll x){
    vector<pll> factors;
    for(ll i = 2; i * i <= x; i++){
        if(x % i == 0){
            ll a = 0;
            for(; x % i == 0; x /= i, a++);
            factors.pb(make_pair(i, a));
        }
        if(x > 1) factors.push_back(make_pair(x, 1));
    }
    return factors;
}
```

6.16 Theorem

6.16.1 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 $|\det(\tilde{L}_{rr})|$.

6.16.2 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 .

6.16.3 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 $T_{n,k} = kn^{n-k-1}$.

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*(2^26)+1, 469762049, 3
//7*17*(2^23)+1, 998244353, 3
//619*202879*(2^3)+1, 1004656809, 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
        if (a[0] < 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 _n) : vector<LL>(_n) {
        copy_n(p.data(), min(p.n(), _n), data());
    }
    Poly& irev() { return reverse(data(), data() + n()), *this; }
    Poly& isz(int _n) { return resize(_n), *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 _n = 1;
        while (_n < n() + rhs.n() - 1) _n <= 1;
        Poly X(*this, _n), Y(rhs, _n);
        ntt(X.data(), _n), ntt(Y.data(), _n);
        fi(0, _n) X[i] = X[i] * Y[i] % P;
        ntt(X.data(), _n, true);
        return X.isz(n() + rhs.n() - 1);
    }
    Poly Inv() const { // (*this)[0] != 0
        if (n() == 1) return {ntt.minv((*this)[0])};
        int _n = 1;
        while (_n < n() * 2) _n <= 1;
        Poly Xi = Poly(*this, (n() + 1) / 2).Inv().isz(_n);
        Poly Y(*this, _n);
        ntt(Xi.data(), _n), ntt(Y.data(), _n);
        fi(0, _n) {
            Xi[i] *= (2 - Xi[i] * Y[i]) % P;
            if ((Xi[i] % P) < 0) Xi[i] += P;
        }
        ntt(Xi.data(), _n, true);
        return Xi.isz(n());
    }
    Poly Sqrt() const { // Jacobi((*this)[0], P) = 1
        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 _n = n() - rhs.n() + 1;
        Poly X(rhs); X.irev().isz(_n);
        Poly Y(*this); Y.irev().isz(_n);
        Poly Q = Y.Mul(X.Inv()).isz(_n).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 _n = (int)x.size();
        if (!_n) return {};
        vector<Poly> down(_n * 2);
        down[1] = DivMod(up[1]).second;
        fi(2, _n * 2) down[i] = down[i / 2].DivMod(up[i]).second;
        /* down[1] = Poly(up[1]).irev().isz(n()).Inv().irev()._tmul(_n, *this);
        fi(2, _n * 2) down[i] = up[i ^ 1]._tmul(up[i].n() - 1, down[i / 2]); */
        vector<LL> y(_n);
        fi(0, _n) y[i] = down[_n + i][0];
        return y;
    }
    static vector<Poly> _tree1(const vector<LL> &x) {
        const int _n = (int)x.size();
        vector<Poly> up(_n * 2);
        fi(0, _n) up[_n + i] = {(x[i] ? P - x[i] : 0), 1};
        for (int i = _n - 1; i > 0; --i) up[i] = up[i * 2].Mul(up[i * 2 + 1]);
        return up;
    }
    vector<LL> Eval(const vector<LL> &x) const {
        auto up = _tree1(x); return _eval(x, up);
    }
    static Poly Interpolate(const vector<LL> &x, const vector<LL> &y) {
        const int _n = (int)x.size();
        vector<Poly> up = _tree1(x), down(_n * 2);
        vector<LL> z = up[1].Dx()._eval(x, up);
        fi(0, _n) z[i] = y[i] * ntt.minv(z[i]) % P;
        fi(0, _n) down[_n + i] = {z[i]};
        for (int i = _n - 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
        return Dx().Mul(Inv()).Sx().isz(n());
    }
    Poly Exp() const { // (*this)[0] == 0
        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());
    }
    Poly Pow(const string &K) const {
        int nz = 0;
        while (nz < n() && !(*this)[nz]) ++nz;
        LL nk = 0, nk2 = 0;
        for (char c : K) {
            nk = (nk * 10 + c - '0') % P;
            nk2 = nk2 * 10 + c - '0';
            if (nk2 * nz >= n()) return Poly(n());
            nk2 %= P - 1;
        }
        if (!nk && !nk2) return Poly(Poly {1}, n());
        Poly X(data() + nz, data() + nz + n() - nz * nk2);
        LL x0 = X[0];
        return X.imul(ntt.minv(x0)).Ln().imul(nk).Exp().imul(ntt.mpow(x0, nk2)).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 = {};

```

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+(const pdd &a, const pdd &b)
{ return pdd(a.X + b.X, a.Y + b.Y);}
pdd operator-(const pdd &a, const pdd &b)
{ return pdd(a.X - b.X, a.Y - b.Y);}
pdd operator*(const pdd &a, const double &b)
{ return pdd(a.X * b, a.Y * b);}
pdd operator/(const pdd &a, const double &b)
{ return pdd(a.X / b, a.Y / b);}
double dot(const pdd &a, const pdd &b)
{ return a.X * b.X + a.Y * b.Y;}
double cross(const pdd &a, const pdd &b)
{ return a.X * b.Y - a.Y * b.X;}
double abs2(const pdd &a)
{ return dot(a, a);}
double abs(const pdd &a)
{ return sqrt(dot(a, a));}
int sign(const double &a)
{ return fabs(a) < eps ? 0 : a > 0 ? 1 : -1;}
int ori(const pdd &a, const pdd &b, const pdd &c)
{ return sign(cross(b - a, c - a));}
bool collinearity(const pdd &p1, const pdd &p2, const
pdd &p3)
{ return fabs(cross(p1 - p3, p2 - p3)) < eps;}
bool btw(const pdd &p1, const pdd &p2, const pdd &p3) {
    if(!collinearity(p1, p2, p3)) return 0;
    return dot(p1 - p3, p2 - p3) < eps;
}
bool seg_intersect(const pdd &p1, const pdd &p2, const
pdd &p3, const 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(const pdd &p1, const pdd &p2, const pdd &
p3, const pdd &p4) {
    double a123 = cross(p2 - p1, p3 - p1);
    double a124 = cross(p2 - p1, p4 - p1);
    return (p4 * a123 - p3 * a124) / (a123 - a124);
}
pdd perp(const pdd &p1)
{ return pdd(-p1.Y, p1.X);}
pdd foot(const pdd &p1, const pdd &p2, const pdd &p3)
{ return intersect(p1, p2, p3, p3 + perp(p2 - p1));}

```

8.2 Convex hull*

```

void hull(vector<p11> &dots) {
    sort(dots.begin(), dots.end());
    vector<p11> 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);
}

```

8.3 External bisector

```

pdd external_bisector(pdd p1, pdd p2, pdd p3){//213
    pdd L1=p2-p1, L2=p3-p1;
    L2=L2*abs(L1)/abs(L2);
    return L1+L2;
}

```

8.4 Heart

```

pdd excenter(pdd p0, pdd p1, pdd p2, double &radius){
    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 radius=abs(center), center+p0;
}

pdd incenter(pdd p1, pdd p2, pdd p3, double &radius){
    double a=abs(p2-p1), b=abs(p3-p1), c=abs(p3-p2);
    double s=(a+b+c)/2, area=sqrt(s*(s-a)*(s-b)*(s-c));
    pdd L1=external_bisector(p1, p2, p3), L2=
        external_bisector(p2, p1, p3);
    return radius=area/s, intersect(p1, p1+L1, p2, p2+L2),
}

pdd escenter(pdd p1, pdd p2, pdd p3){//213
    pdd L1=external_bisector(p1, p2, p3), L2=
        external_bisector(p2, p2+p2-p1, p3);
    return intersect(p1, p1+L1, p2, p2+L2);
}

pdd barycenter(pdd p1, pdd p2, pdd p3){
    return (p1+p2+p3)/3;
}

pdd orthocenter(pdd p1, pdd p2, pdd p3){
    pdd L1=p3-p2, L2=p3-p1;
    swap(L1.X, L1.Y), L1.X*=-1;
    swap(L2.X, L2.Y), L2.X*=-1;
    return intersect(p1, p1+L1, p2, p2+L2);
}

```

8.5 Minimum Circle Cover*

```

pdd Minimum_Circle_Cover(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;
}

```

8.6 Polar Angle Sort*

```
pdd center;//sort base
int Quadrant(pdd a) {
    if(a.X > 0 && a.Y >= 0) return 1;
    if(a.X <= 0 && a.Y > 0) return 2;
    if(a.X < 0 && a.Y <= 0) return 3;
    if(a.X >= 0 && a.Y < 0) return 4;
}
bool cmp(p11 a, p11 b) {
    a = a - center, b = b - center;
    if (Quadrant(a) != Quadrant(b))
        return Quadrant(a) < Quadrant(b);
    if (cross(b, a) == 0) return abs2(a) < abs2(b);
    return cross(a, b) > 0;
}
bool cmp(pdd a, pdd b) {
    a = a - center, b = b - center;
    if(fabs(atan2(a.Y, a.X) - atan2(b.Y, b.X)) > eps)
        return atan2(a.Y, a.X) < atan2(b.Y, b.X);
    return abs(a) < abs(b);
}
```

8.7 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;
}
```

8.8 Intersection of polygon and circle

```
// Divides into multiple triangle, and sum up
// test by HDU2892
const double PI=acos(-1);
double _area(pdd pa, pdd pb, double r){
    if(abs(pa)<abs(pb)) swap(pa, pb);
    if(abs(pb)<eps) return 0;
    double S, h, theta;
    double a=abs(pb),b=abs(pa),c=abs(pb-pa);
    double cosB = dot(pb,pb-pa) / a / c, B = acos(cosB);
    double cosC = dot(pa,pb) / a / b, C = acos(cosC);
    if(a > r){
        S = (C/2)*r*r;
        h = a*b*sin(C)/c;
        if (h < r && B < PI/2) S -= (acos(h/r)*r*r - h*sqrt
            (r*r-h*h));
    }
    else if(b > r){
        theta = PI - B - asin(sin(B)/r*a);
        S = .5*a*r*sin(theta) + (C-theta)/2*r*r;
    }
    else S = .5*sin(C)*a*b;
    return S;
}
double area_poly_circle(const vector<pdd> poly,const
    pdd &O,const double r){
    double S=0;
    for(int i=0;i<SZ(poly);++i)
        S+=_area(poly[i]-O,poly[(i+1)%SZ(poly)]-O,r)*ori(0,
            poly[i],poly[(i+1)%SZ(poly)]);
    return fabs(S);
}
```

8.9 Intersection of line and circle

```
vector<pdd> line_interCircle(const pdd &p1,const pdd &
    p2,const pdd &c,const double r){
    pdd ft=foot(p1,p2,c),vec=p2-p1;
    double dis=abs(c-ft);
    if(fabs(dis-r)<eps) return vector<pdd>{ft};
    if(dis>r) return {};
    vec=vec*sqrt(r*r-dis*dis)/abs(vec);
    return vector<pdd>{ft+vec,ft-vec};
}
```

8.10 point in circle

```
// return p4 is strictly in circumcircle of tri(p1,p2,
    p3)
long long sqr(long long x) { return x * x; }
bool in_cc(const p11& p1, const p11& p2, const p11& p3,
    const p11& p4) {
    long long u11 = p1.X - p4.X; long long u12 = p1.Y -
        p4.Y;
    long long u21 = p2.X - p4.X; long long u22 = p2.Y -
        p4.Y;
    long long u31 = p3.X - p4.X; long long u32 = p3.Y -
        p4.Y;
    long long u13 = sqr(p1.X) - sqr(p4.X) + sqr(p1.Y) -
        sqr(p4.Y);
    long long u23 = sqr(p2.X) - sqr(p4.X) + sqr(p2.Y) -
        sqr(p4.Y);
    long long 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;
}
```

8.11 Half plane intersection

```
bool isin( Line l0, Line l1, Line l2 ){
    // Check inter(l1, l2) in l0
    pdd p = intersect(l1.X,l1.Y,l2.X,l2.Y);
    return cross(l0.Y - l0.X,p - l0.X) > eps;
}
/* If no solution, check: 1. ret.size() < 3
 * Or more precisely, 2. interPnt(ret[0], ret[1])
 * in all the lines. (use (L.Y - L.X) ^ (p - L.X) > 0
 */
/* --^-- Line.X --^-- Line.Y --^-- */
vector<Line> halfPlaneInter(vector<Line> lines){
    int sz = lines.size();
    vector<double> ata(sz),ord(sz);
    for(int i=0; i<sz; ++i) {
        ord[i] = i;
        pdd d = lines[i].Y - lines[i].X;
        ata[i] = atan2(d.Y, d.X);
    }
    sort(ord.begin(), ord.end(), [&](int i,int j){
        if( fabs(ata[i] - ata[j]) < eps )
            return (cross(lines[i].Y-lines[i].X,
                lines[j].Y-lines[j].X))<0;
        return ata[i] < ata[j];
    });
    vector<Line> fin;
    for (int i=0; i<sz; ++i)
        if (!i || fabs(ata[ord[i]] - ata[ord[i-1]]) > eps)
            fin.pb(lines[ord[i]]);
    deque<Line> dq;
    for (int i=0; i<SZ(fin); i++){
        while(SZ(dq)>=2&&!isin(fin[i],dq[SZ(dq)-2],dq.back
            ()))
            dq.pop_back();
        while(SZ(dq)>=2&&!isin(fin[i],dq[0],dq[1]))
            dq.pop_front();
        dq.push_back(fin[i]);
    }
}
```



```

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();
vector<Line> res(ALL(dq));
return res;
}

```

8.12 CircleCover*

```

const int N = 1021;
struct CircleCover {
    int C;
    Cir c[N];
    bool g[N][N], overlap[N][N];
    // Area[i] : area covered by at least i circles
    double Area[ N ];
    void init(int _C){ C = _C;}
    struct Teve {
        pdd p; double ang; int add;
        Teve() {}
        Teve(pdd _a, double _b, int _c):p(_a), ang(_b), add
            (_c){}
        bool operator<(const Teve &a)const
        {return ang < a.ang;}
    }eve[N * 2];
    // strict: x = 0, otherwise x = -1
    bool disjuct(Cir &a, Cir &b, int x)
    {return sign(abs(a.O - b.O) - a.R - b.R) > x;}
    bool contain(Cir &a, Cir &b, int x)
    {return sign(a.R - b.R - abs(a.O - b.O)) > x;}
    bool contain(int i, int j) {
        /* c[j] is non-strictly in c[i]. */
        return (sign(c[i].R - c[j].R) > 0 || (sign(c[i].R -
            c[j].R) == 0 && i < j)) && contain(c[i], c[j],
            -1);
    }
    void solve(){
        fill_n(Area, C + 2, 0);
        for(int i = 0; i < C; ++i)
            for(int j = 0; j < C; ++j)
                overlap[i][j] = contain(i, j);
        for(int i = 0; i < C; ++i)
            for(int j = 0; j < C; ++j)
                g[i][j] = !(overlap[i][j] || overlap[j][i] ||
                    disjuct(c[i], c[j], -1));
        for(int i = 0; i < C; ++i){
            int E = 0, cnt = 1;
            for(int j = 0; j < C; ++j)
                if(j != i && overlap[j][i])
                    ++cnt;
            for(int j = 0; j < C; ++j)
                if(i != j && g[i][j]) {
                    pdd aa, bb;
                    CCinter(c[i], c[j], aa, bb);
                    double A = atan2(aa.Y - c[i].O.Y, aa.X - c[i]
                        .O.X);
                    double B = atan2(bb.Y - c[i].O.Y, bb.X - c[i]
                        .O.X);
                    eve[E++] = Teve(bb, B, 1), eve[E++] = Teve(aa
                        , A, -1);
                    if(B > A) ++cnt;
                }
            if(E == 0) Area[cnt] += pi * c[i].R * c[i].R;
            else{
                sort(eve, eve + E);
                eve[E] = eve[0];
                for(int j = 0; j < E; ++j){
                    cnt += eve[j].add;
                    Area[cnt] += cross(eve[j].p, eve[j + 1].p) *
                        .5;
                    double theta = eve[j + 1].ang - eve[j].ang;
                    if (theta < 0) theta += 2. * pi;
                    Area[cnt] += (theta - sin(theta)) * c[i].R *
                        c[i].R * .5;
                }
            }
        }
    }
};

```

8.13 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 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);}

```

8.14 Convexhull3D*

```

struct CH3D {
    struct face{int a, b, c; bool ok;} F[8 * N];
    double dblcmp(Point &p, face &f)
    {return dot(cross3(P[f.a], P[f.b], P[f.c]), p - P[f.a]
        );}
    int g[N][N], num, n;
    Point P[N];
    void deal(int p, int a, int b) {
        int f = g[a][b];
        face add;
        if (F[f].ok) {
            if (dblcmp(P[p], F[f]) > eps) dfs(p, f);
            else
                add.a = b, add.b = a, add.c = p, add.ok = 1, g[
                    p][b] = g[a][p] = g[b][a] = num, F[num++] =
                    add;
        }
    }
    void dfs(int p, int now) {
        F[now].ok = 0;
        deal(p, F[now].b, F[now].a), deal(p, F[now].c, F[
            now].b), deal(p, F[now].a, F[now].c);
    }
    bool same(int s, int t){
        Point &a = P[F[s].a];
        Point &b = P[F[s].b];
        Point &c = P[F[s].c];
        return fabs(volume(a, b, c, P[F[t].a])) < eps &&
            fabs(volume(a, b, c, P[F[t].b])) < eps && fabs(
                volume(a, b, c, P[F[t].c])) < eps;
    }
    void init(int _n){n = _n, num = 0;}
    void solve() {
        face add;
        num = 0;
        if(n < 4) return;
        if([&](){
            for (int i = 1; i < n; ++i)
                if (abs(P[0] - P[i]) > eps)
                    return swap(P[1], P[i]), 0;
            return 1;
        }) || [&](){
            for (int i = 2; i < n; ++i)
                if (abs(cross3(P[i], P[0], P[1])) > eps)
                    return swap(P[2], P[i]), 0;
            return 1;
        }) || [&](){
            for (int i = 3; i < n; ++i)
                if (fabs(dot(cross(P[0] - P[1], P[1] - P[2]), P
                    [0] - P[i])) > eps)
                    return swap(P[3], P[i]), 0;
            return 1;
        }
    }
};

```

```

    }())return;
    for (int i = 0; i < 4; ++i) {
        add.a = (i + 1) % 4, add.b = (i + 2) % 4, add.c =
            (i + 3) % 4, add.ok = true;
        if (dblcmp(P[i], add) > 0) swap(add.b, add.c);
        g[add.a][add.b] = g[add.b][add.c] = g[add.c][add.
            a] = num;
        F[num++] = add;
    }
    for (int i = 4; i < n; ++i)
        for (int j = 0; j < num; ++j)
            if (F[j].ok && dblcmp(P[i], F[j]) > eps) {
                dfs(i, j);
                break;
            }
    for (int tmp = num, i = (num = 0); i < tmp; ++i)
        if (F[i].ok) F[num++] = F[i];
}
double get_area() {
    double res = 0.0;
    if (n == 3)
        return abs(cross3(P[0], P[1], P[2])) / 2.0;
    for (int i = 0; i < num; ++i)
        res += area(P[F[i].a], P[F[i].b], P[F[i].c]);
    return res / 2.0;
}
double get_volume() {
    double res = 0.0;
    for (int i = 0; i < num; ++i)
        res += volume(Point(0, 0, 0), P[F[i].a], P[F[i].b
            ], P[F[i].c]);
    return fabs(res / 6.0);
}
int triangle() {return num;}
int polygon() {
    int res = 0;
    for (int i = 0, flag = 1; i < num; ++i, res += flag
        , flag = 1)
        for (int j = 0; j < i && flag; ++j)
            flag &= !same(i, j);
    return res;
}
Point getcent(){
    Point ans(0, 0, 0), temp = P[F[0].a];
    double v = 0.0, t2;
    for (int i = 0; i < num; ++i)
        if (F[i].ok == true) {
            Point p1 = P[F[i].a], p2 = P[F[i].b], p3 = P[F[
                i].c];
            t2 = volume(temp, p1, p2, p3) / 6.0;
            if (t2 > 0)
                ans.x += (p1.x + p2.x + p3.x + temp.x) * t2,
                ans.y += (p1.y + p2.y + p3.y + temp.y) *
                    t2, ans.z += (p1.z + p2.z + p3.z + temp.z
                        ) * t2, v += t2;
        }
    ans.x /= (4 * v), ans.y /= (4 * v), ans.z /= (4 * v
        );
    return ans;
}
double pointmindis(Point p) {
    double rt = 99999999;
    for (int i = 0; i < num; ++i)
        if (F[i].ok == true) {
            Point p1 = P[F[i].a], p2 = P[F[i].b], p3 = P[F[
                i].c];
            double a = (p2.y - p1.y) * (p3.z - p1.z) - (p2.
                z - p1.z) * (p3.y - p1.y);
            double b = (p2.z - p1.z) * (p3.x - p1.x) - (p2.
                x - p1.x) * (p3.z - p1.z);
            double c = (p2.x - p1.x) * (p3.y - p1.y) - (p2.
                y - p1.y) * (p3.x - p1.x);
            double d = 0 - (a * p1.x + b * p1.y + c * p1.z
                );
            double temp = fabs(a * p.x + b * p.y + c * p.z
                + d) / sqrt(a * a + b * b + c * c);
            rt = min(rt, temp);
        }
    return rt;
}
};

```

8.15 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.push_back(now);
    for (int i = 0; i < now->num_chd(); ++i)
        go(now -> chd[i]);
}
void build(int n, pll* ps) { // build triangulation
    tris = pool; triang.clear(); vst.clear();
    random_shuffle(ps, ps + n);
    Trig tri; // the triangulation structure
    for (int i = 0; i < n; ++i)
        tri.add_point(ps[i]);
    go(tri.the_root);
}

```

8.16 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));
            }
    }
}

```

8.17 Tangent line of two circles

```

vector<Line> go(const Cir& c1, const Cir& c2, int sign1) {
    // sign1 = 1 for outer tang, -1 for inter tang
    vector<Line> ret;
    double d_sq = norm2(c1.O - c2.O);
    if (d_sq < eps) return ret;
    double d = sqrt(d_sq);
    Pt v = (c2.O - c1.O) / d;
    double c = (c1.R - sign1 * c2.R) / d;
    if (c * c > 1) return ret;
    double h = sqrt(max(0.0, 1.0 - c * c));
    for (int sign2 = 1; sign2 >= -1; sign2 -= 2) {
        Pt n = { v.X * c - sign2 * h * v.Y, v.Y * c + sign2 * h * v.X };
        Pt p1 = c1.O + n * c1.R;
        Pt p2 = c2.O + n * (c2.R * sign1);
        if (fabs(p1.X - p2.X) < eps and fabs(p1.Y - p2.Y) < eps)
            p2 = p1 + perp(c2.O - c1.O);
        ret.push_back({ p1, p2 });
    }
    return ret;
}

```

8.18 minMaxEnclosingRectangle

```

pdd solve(vector<pll> &dots) {
    vector<pll> hull;
    const double INF=1e18, qi=acos(-1)/2*3;
    cv.dots=dots;
    hull=cv.hull();
    double Max=0, Min=INF, deg;
    ll n=hull.size();
    hull.pb(hull[0]);
    for (int i=0, u=1, r=1, l; i<n; ++i) {
        pll nw=hull[i+1]-hull[i];
        while (cross(nw, hull[u+1]-hull[i]) > cross(nw, hull[u]-hull[i]))
            u=(u+1)%n;
        while (dot(nw, hull[r+1]-hull[i]) > dot(nw, hull[r]-hull[i]))
            r=(r+1)%n;
        if (!i) l=(r+1)%n;
        while (dot(nw, hull[l+1]-hull[i]) < dot(nw, hull[l]-hull[i]))
            l=(l+1)%n;
        Min=min(Min, (double)(dot(nw, hull[r]-hull[i]) - dot(nw, hull[l]-hull[i])) * cross(nw, hull[u]-hull[i]) / abs2(nw));
        deg=acos((double)dot(hull[r]-hull[l], hull[u]-hull[i]) / (abs(hull[r]-hull[l]) * abs(hull[u]-hull[i])));
        deg=(qi-deg)/2;
        Max=max(Max, (double)abs(hull[r]-hull[l]) * abs(hull[u]-hull[i]) * sin(deg) * sin(deg));
    }
    return pdd(Min, Max);
}

```

8.19 minDistOfTwoConvex

```
// p, q is convex
double TwoConvexHullMinDist(Point P[], Point Q[], int n
, int m) {
    int YMinP = 0, YMaxQ = 0;
    double tmp, ans = 999999999;
    for (i = 0; i < n; ++i) if (P[i].y < P[YMinP].y) YMinP = i;
    for (i = 0; i < m; ++i) if (Q[i].y > Q[YMaxQ].y) YMaxQ = i;
    P[n] = P[0], Q[m] = Q[0];
    for (int i = 0; i < n; ++i) {
        while (tmp = Cross(Q[YMaxQ + 1] - P[YMinP + 1], P[
            YMinP] - P[YMinP + 1]) > Cross(Q[YMaxQ] - P[
            YMinP + 1], P[YMinP] - P[YMinP + 1])) YMaxQ = (
            YMaxQ + 1) % m;
        if (tmp < 0) ans = min(ans, PointToSegDist(P[YMinP
            ], P[YMinP + 1], Q[YMaxQ]));
        else ans = min(ans, TwoSegMinDist(P[YMinP], P[YMinP
            + 1], Q[YMaxQ], Q[YMaxQ + 1]));
        YMinP = (YMinP + 1) % n;
    }
    return ans;
}
```

8.20 Minkowski Sum*

```
vector<p11> Minkowski(vector<p11> A, vector<p11> B) {
    hull(A), hull(B);
    vector<p11> 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 p1 = 0, p2 = 0; p1 < SZ(A) || p2 < SZ(B);)
        if (p2 >= SZ(B) || (p1 < SZ(A) && cross(s1[p1], s2[
            p2]) >= 0))
            C.pb(C.back() + s1[p1++]);
        else
            C.pb(C.back() + s2[p2++]);
    return hull(C), C;
}
```

8.21 RotatingSweepLine

```
void rotatingSweepLine(vector<pii> &ps) {
    int n = SZ(ps);
    vector<int> id(n), pos(n);
    vector<pii> line(n * (n - 1) / 2);
    int m = 0;
    for (int i = 0; i < n; ++i)
        for (int j = i + 1; j < n; ++j)
            line[m++] = pii(i, j);
    sort(ALL(line), [&](const pii &a, const pii &b) ->
        bool {
            if (ps[a.X].X == ps[a.Y].X)
                return 0;
            if (ps[b.X].X == ps[b.Y].X)
                return 1;
            return (double)(ps[a.X].Y - ps[a.Y].Y) / (ps[a.X
                ].X - ps[a.Y].X) < (double)(ps[b.X].Y - ps[b
                .Y].Y) / (ps[b.X].X - ps[b.Y].X);
        });
    iota(id, id + n, 0);
    sort(ALL(id), [&](const int &a, const int &b){ return
        ps[a] < ps[b]; });
    for (int i = 0; i < n; ++i) pos[id[i]] = i;
    for (int i = 0; i < m; ++i) {
        auto l = line[i];
        // meow
        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
            );
    }
}
```

8.22 Simpson

```
const double eps = 1e-9;
double F(double x){
}
double Simpson(double l, double r){
    return (r-l)*(F(r)+F(l)+4.0*F((r+l)*0.5))/6.0;
}
double simpson(double l, double r, double cur){
    double mid = (l+r)*0.5;
    double L = Simpson(l, mid), R = Simpson(mid, r);
    if (fabs(cur-L-R) <= eps)
        return L+R;
    else
        return simpson(l, mid, L)+simpson(mid, r, R);
}
double simpson_ada(double l, double r, double cur, double
    e){
    double mid = (l+r)*0.5;
    double L = Simpson(l, mid), R = Simpson(mid, r);
    if (fabs(cur-L-R) <= e)
        return L+R;
    else
        return simpson(l, mid, L, e*0.5)+simpson(mid, r, R, e
            *0.5);
}
```

9 Else

9.1 Mo's Alogrithm(With modification)

```
struct QUERY{//BLOCK=N^{2/3}
    int L,R,id,LBId,RBId,T;
    QUERY(int l,int r,int id,int lb,int rb,int t):
        L(l),R(r),id(id),LBId(lb),RBId(rb),T(t){}
    bool operator<(const QUERY &b)const{
        if (LBId!=b.LBId) return LBId<b.LBId;
        if (RBId!=b.RBId) return RBId<b.RBId;
        return T<b.T;
    }
};
vector<QUERY> query;
int cur_ans, arr[MAXN], ans[MAXN];
void addTime(int L,int R,int T){}
void subTime(int L,int R,int T){}
void add(int x){}
void sub(int x){}
void solve(){
    sort(ALL(query));
    int L=0,R=0,T=-1;
    for(auto q:query){
        while(T<q.T) addTime(L,R,++T);
        while(T>q.T) subTime(L,R,T--);
        while(R<q.R) add(arr[++R]);
        while(L>q.L) add(arr[--L]);
        while(R>q.R) sub(arr[R--]);
        while(L<q.L) sub(arr[L++]);
        ans[q.id]=cur_ans;
    }
}
```

9.2 Mo's Alogrithm On Tree

```
const int MAXN=40005;
vector<int> G[MAXN]; //1-base
int n,B,arr[MAXN],ans[100005],cur_ans;
int in[MAXN],out[MAXN],dfn[MAXN*2],dft;
int deep[MAXN],sp[___lg(MAXN*2)+1][MAXN*2],bln[MAXN],spt
    ;
bitset<MAXN> inset;
struct QUERY{
    int L,R,Lid,id,lca;
    QUERY(int l,int r,int _id):L(l),R(r),lca(0),id(_id){}
    bool operator<(const QUERY &b){
        if (Lid!=b.Lid) return Lid<b.Lid;
    }
}
```

```

    return R<b.R;
}
};
vector<QUERY> query;
void dfs(int u, int f, int d){
    deep[u]=d, sp[0][spt]=u, bln[u]=spt++;
    dfn[dft]=u, in[u]=dft++;
    for(int v:G[u])
        if(v!=f)
            dfs(v, u, d+1), sp[0][spt]=u, bln[u]=spt++;
    dfn[dft]=u, out[u]=dft++;
}
int lca(int u, int v){
    if(bln[u]>bln[v]) swap(u, v);
    int t=__lg(bln[v]-bln[u]+1);
    int a=sp[t][bln[u]], b=sp[t][bln[v]-(1<<t)+1];
    if(deep[a]<deep[b]) return a;
    return b;
}
void sub(int x){}
void add(int x){}
void flip(int x){
    if(inset[x]) sub(arr[x]);
    else add(arr[x]);
    inset[x]=~inset[x];
}
void solve(){
    B=sqrt(2*n), dft=spt=cur_ans=0, dfs(1, 1, 0);
    for(int i=1, x=2; x<2*n; ++i, x<=1)
        for(int j=0; j+x<2*n; ++j)
            if(deep[sp[i-1][j]]<deep[sp[i-1][j+x/2]])
                sp[i][j]=sp[i-1][j];
            else sp[i][j]=sp[i-1][j+x/2];
    for(auto &q:query){
        int c=lca(q.L, q.R);
        if(c==q.L || c==q.R)
            q.L=out[c==q.L?q.R:q.L], q.R=out[c];
        else if(out[q.L]<in[q.R])
            q.lca=c, q.L=out[q.L], q.R=in[q.R];
        else q.lca=c, c=in[q.L], q.L=out[q.R], q.R=c;
        q.Lid=q.L/B;
    }
    sort(ALL(query));
    int L=0, R=-1;
    for(auto q:query){
        while(R<q.R) flip(dfn[++R]);
        while(L>q.L) flip(dfn[--L]);
        while(R>q.R) flip(dfn[R--]);
        while(L<q.L) flip(dfn[L++]);
        if(q.lca) add(arr[q.lca]);
        ans[q.id]=cur_ans;
        if(q.lca) sub(arr[q.lca]);
    }
}
}

```

```

    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;
}
};

```

9.4 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.

9.3 DynamicConvexTrick*

```

// only works for integer coordinates!!
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);
    }
};

```