

Contents

1	Basic	
1.1	default	1
1.2	vimrc	1
1.3	judge	1
2	Matching and Flow	1
2.1	Dinic	1
2.2	zkwDinic	1
2.3	HopcroftKarp	2
2.4	KM	2
2.5	SW	2
2.6	GeneralMatching	2
3	Graph	3
3.1	2-SAT	3
3.2	ManhattanMST	3
4	Data Structure	3
4.1	Lazy Segtree	3
4.2	LiChao Segtree	3
4.3	Persistent SegmentTree	4
4.4	Blackmagic	4
4.5	Centroid Decomposition	4
5	Math	4
5.1	Theorem	4
5.2	Exgcd	4
5.3	CRT	5
5.4	Factorize	5
5.5	NTT	5
5.6	FWT	6
5.7	FWT	6
5.8	Lucas	6
5.9	Berlekamp Massey	6
5.10	Gauss Elimination	6
5.11	Linear Equation	6
5.12	LinearRec	7
5.13	SubsetConv	7
5.14	FloorSum	7
6	Geometry	7
6.1	2D Point	7
6.2	Convex Hull	8
6.3	Convex Hull trick	8
6.4	Dynamic Convex Hull	8
6.5	Half Plane Intersection	8
6.6	Minimal Enclosing Circle	8
6.7	Minkowski	8
7	Stringology	9
7.1	Z-algorithm	9
7.2	Manacher	9
7.3	SuffixArray	9
7.4	PalindromicTree	9
7.5	SmallestRotation	10
8	Misc	10
8.1	HilbertCurve	10
8.2	DLX	10

1 Basic

1.1 default

```
#include <bits/stdc++.h>
using namespace std;
#ifdef LOCAL
template<class ...T> void dbg(T ...x) { char e{}; ((
    cerr << e << x, e = ' ', ...); }
template<class T> void org(T l, T r) { while (l != r)
    cerr << ' ' << *l++; cerr << '\n'; }
#define debug(x...) dbg("(", #x, ") =", x, '\n')
#define orange(x...) dbg("[", #x, "] =", org(x))
#else
#pragma GCC optimize("O3,unroll-loops")
#pragma GCC target("avx2,bmi,bmi2,lzcnt,popcnt")
#define debug(...) ((void)0)
#define orange(...) ((void)0)
#endif
#define ff first
#define ss second
#define all(v) (v).begin(), (v).end()
#define rall(v) (v).rbegin(), (v).rend()
template<class T> bool chmin(T &a, T b) { return b < a
    and (a = b, true); }
template<class T> bool chmax(T &a, T b) { return a < b
    and (a = b, true); }
```

1.2 vimrc

```
set ts=4 sw=4 nu rnu et cin hls mouse=a
color default
sy on
inoremap {<CR> {<CR>}<C-o>0
inoremap jk <Esc>
nnoremap J 5j
nnoremap K 5k
nnoremap <F8> :w<bar>!g++ -std=c++20 -DLOCAL -Wfatal-
    errors -Wshadow -O2 -g -fsanitize=address,undefined
    -o run "%" && echo "done." && time ./run<CR>
```

1.3 judge

```
set -e
g++ -O3 a.cpp -o a
g++ -O3 ac.cpp -o c
g++ -O3 gen.cpp -o g
for ((i=0;;i++))
do
    echo "case $i"
    ./g > inp
    time ./a < inp > wa.out
    time ./c < inp > ac.out
    diff ac.out wa.out || break
done
```

2 Matching and Flow

2.1 Dinic

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

2.2 zkwDinic

```
template<class Cap>
struct zkwDinic {
    struct Edge { int v; Cap w, f; int rev; };
    vector<vector<Edge>> G;
    int n, S, T;
    zkwDinic(int _n, int _S, int _T) : n(_n), S(_S), T(_T), G(_n) {}
    void add_edge(int u, int v, Cap w, Cap f) {
        G[u].push_back({v, w, f, (int)G[v].size()});
        G[v].push_back({u, -w, 0, (int)G[u].size() - 1});
    }
    vector<Cap> dis;
    vector<bool> vis;
    bool spfa() {
        queue<int> que;
        dis.assign(n, INF);
        vis.assign(n, false);
        que.push(S);
        vis[S] = 1;
        dis[S] = 0;
        while (!que.empty()) {
            int u = que.front(); que.pop();
            vis[u] = 0;
            for (auto [v, w, f, _] : G[u])
                if (f and chmin(dis[v], dis[u] + w))
                    if (!vis[v]) que.push(v), vis[v] = 1;
        }
        return dis[T] != INF;
    }
    Cap dfs(int u, Cap in) {
        if (u == T) return in;
        vis[u] = 1;
        Cap out = 0;
        for (auto &[v, w, f, rev] : G[u])
            if (f and !vis[v] and dis[v] == dis[u] + w) {
                Cap x = dfs(v, min(in, f));
                in -= x, out += x;
                f -= x, G[v][rev].f += x;
                if (!in) break;
            }
        if (in) dis[u] = INF;
        vis[u] = 0;
        return out;
    }
    pair<Cap, Cap> maxflow() {
        Cap a = 0, b = 0;
        while (spfa()) {
            Cap x = dfs(S, INF);
            a += x;
            b += x * dis[T];
        }
        return {a, b};
    }
};
```

2.3 HopcroftKarp

```
struct HopcroftKarp {
    std::vector<int> g, l, r;
    int ans;
    HopcroftKarp(int n, int m, const std::vector<pair<int, int>> &e)
        : g(e.size()), l(n, -1), r(m, -1), ans(0) {
        vector<int> deg(n + 1);
        for (auto [x, y] : e) deg[x]++;
        partial_sum(all(deg), deg.begin());
        for (auto [x, y] : e) g[--deg[x]] = y;
        vector<int> que(n);
        for (;;) {
            vector<int> a(n, -1), p(n, -1);
            int t = 0;
            for (int i = 0; i < n; i++) if (l[i] == -1)
                que[t++] = a[i] = p[i] = i;
            bool match = false;
            for (int i = 0; i < t; i++) {
                int x = que[i];
                if (~l[a[x]]) continue;
                for (int j = deg[x]; j < deg[x + 1]; j++) {
                    int y = g[j];
                    if (r[y] == -1) {
                        while (~y) r[y] = x, swap(l[x], y), x = p[x];
                        match = true, ans++;
                        break;
                    }
                }
            }
            if (!match) break;
        }
    }
};
```

```
match = true, ans++;
break;
}
if (p[r[y]] == -1)
    que[t++] = y = r[y], p[y] = x, a[y] = a[x];
}
if (!match) break;
}
}
};
```

2.4 KM

```
i64 KM(vector<vector<int>> W) {
    const int n = W.size();
    vector<int> fl(n, -1), fr(n, -1), hr(n), hl(n);
    for (int i = 0; i < n; ++i) {
        hl[i] = *max_element(W[i].begin(), W[i].end());
    }
    auto Bfs = [&](int s) {
        vector<int> slk(n, INF), pre(n);
        vector<bool> vl(n, false), vr(n, false);
        queue<int> que;
        que.push(s);
        vr[s] = true;
        auto Check = [&](int x) -> bool {
            if (vl[x] == true, fl[x] != -1) {
                que.push(fl[x]);
                return vr[fl[x]] == true;
            }
        };
        while (x != -1) swap(x, fr[fl[x] = pre[x]]);
        return false;
    };
    while (true) {
        while (!que.empty()) {
            int y = que.front(); que.pop();
            for (int x = 0, d = 0; x < n; ++x) {
                if (!vl[x] and slk[x] >= (d = hl[x] + hr[y] - W[x][y])) {
                    if (pre[x] == y, d) slk[x] = d;
                    else if (!Check(x)) return;
                }
            }
        }
        int d = INF;
        for (int x = 0; x < n; ++x) {
            if (!vl[x] and 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] and !slk[x] and !Check(x)) return;
        }
    }
};
for (int i = 0; i < n; ++i) Bfs(i);
i64 res = 0;
for (int i = 0; i < n; ++i) res += W[i][fl[i]];
return res;
```

2.5 SW

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

```

        if (del[i] || v[i]) continue;
        g[i] += w[c][i];
    }
    return make_pair(s, t);
}
int GlobalMinCut(int n) {
    int cut = kInf;
    fill(del, 0, sizeof(del));
    for (int i = 0; i < n - 1; ++i) {
        int s, t; tie(s, t) = Phase(n);
        del[t] = 1, cut = min(cut, g[t]);
        for (int j = 0; j < n; ++j) {
            w[s][j] += w[t][j];
            w[j][s] += w[j][t];
        }
    }
    return cut;
}

```

2.6 GeneralMatching

```

struct GeneralMatching {
    const int BLOCK = 10;
    int n;
    vector<vector<int>> g;
    vector<int> hit, mat;
    std::priority_queue<pair<i64, int>, vector<pair<i64, int>>> unmat;
    GeneralMatching(int _n) : n(_n), g(_n), mat(n, -1), hit(n) {}
    void add_edge(int a, int b) { // 0 <= a != b < n
        g[a].push_back(b);
        g[b].push_back(a);
    }
    int get_match() {
        for (int i = 0; i < n; i++) if (!g[i].empty()) {
            unmat.emplace(0, i);
        }
        // If WA, increase this
        // there are some cases that need >= 1.3*n^2 steps
        for BLOCK=1
        // no idea what the actual bound needed here is.
        const int MAX_STEPS = 10 + 2 * n + n * n / BLOCK / 2;
        mt19937 rng(random_device{}());
        for (int i = 0; i < MAX_STEPS; ++i) {
            if (unmat.empty()) break;
            int u = unmat.top().second;
            unmat.pop();
            if (mat[u] != -1) continue;
            for (int j = 0; j < BLOCK; j++) {
                ++hit[u];
                auto &e = g[u];
                const int v = e[rng() % e.size()];
                mat[u] = v;
                swap(u, mat[v]);
                if (u == -1) break;
            }
            if (u != -1) {
                mat[u] = -1;
                unmat.emplace(hit[u] * 100ULL / (g[u].size() + 1), u);
            }
        }
        int siz = 0;
        for (auto e : mat) siz += (e != -1);
        return siz / 2;
    }
};

```

3 Graph

3.1 2-SAT

```

struct TwoSAT {
    vector<vector<int>> G;
    int n;
    TwoSAT(int _n) : n(_n), G(_n * 2) {}
    int ne(int x) { return x < n ? x + n : x - n; }
    void add_edge(int u, int v) { // u or v
        G[ne(u)].push_back(v);
        G[ne(v)].push_back(u);
    }
};

```

```

}
vector<int> solve() {
    vector<int> ans(n * 2, -1), id(n * 2), stk, \
        low(n * 2), dfn(n * 2), vis(n * 2);
    int _t = 0, scc_cnt = 0;
    function<void(int)> dfs = [&](int u) {
        dfn[u] = low[u] = _t++;
        stk.push_back(u);
        vis[u] = 1;
        for (int v : G[u]) {
            if (!vis[v])
                dfs(v), chmin(low[u], low[v]);
            else if (vis[v] == 1)
                chmin(low[u], dfn[v]);
        }
        if (dfn[u] == low[u]) {
            for (int x = -1; x != u; ) {
                x = stk.back(); stk.pop_back();
                vis[x] = 2, id[x] = scc_cnt;
                if (ans[x] == -1) {
                    ans[x] = 1;
                    ans[ne(x)] = 0;
                }
            }
            scc_cnt++;
        }
    };
    for (int i = 0; i < n + n; i++)
        if (!vis[i]) dfs(i);
    for (int i = 0; i < n; i++)
        if (id[i] == id[ne(i)])
            return {};
    ans.resize(n);
    return ans;
}

```

3.2 ManhattanMST

```

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

```

4 Data Structure

4.1 Lazy Segtree

```

template<class S, class T>
struct Seg {
    Seg<S, T> *ls{}, *rs{};
    int l, r;
    S d{};
    T f{};
    Seg(int _l, int _r, const vector<Info> &v) : l{_l}, r
        {_r} {
        if (r - l == 1) {
            d = v[l];
            return;
        }
    }
};

```

```

    int mid = 1 + r >> 1;
    ls = new Seg(1, mid, v);
    rs = new Seg(mid, r, v);
    pull();
}
void upd(const T &g) {
    g(d), g(f);
}
void pull() {
    d = ls->d + rs->d;
}
void push() {
    ls->upd(f);
    rs->upd(f);
    f = T{};
}
S prod(int x, int y) {
    if (y <= 1 or r <= x) return S{};
    if (x <= 1 and r <= y) return d;
    push();
    return ls->prod(x, y) + rs->prod(x, y);
}
void apply(int x, int y, const T &g) {
    if (y <= 1 or r <= x) return;
    if (x <= 1 and r <= y) {
        upd(g);
        return;
    }
    push();
    ls->apply(x, y, g);
    rs->apply(x, y, g);
    pull();
}
};

```

4.2 LiChao Segtree

```

struct Line {
    i64 k, m; // y = k + mx;
    Line() : k{INF}, m{} {}
    Line(i64 _k, i64 _m) : k(_k), m(_m) {}
    i64 get(i64 x) {
        return k + m * x;
    }
};
struct Seg {
    Seg *ls{}, *rs{};
    int l, r, mid;
    Line line{};
    Seg(int _l, int _r) : l(_l), r(_r), mid(_l + _r >> 1) {
        if (r - l == 1) return;
        ls = new Seg(1, mid);
        rs = new Seg(mid, r);
    }
    void insert(Line L) {
        if (line.get(mid) > L.get(mid))
            swap(line, L);
        if (r - l == 1) return;
        if (L.m < line.m) {
            rs->insert(L);
        } else {
            ls->insert(L);
        }
    }
    i64 query(int p) {
        if (p < l or r <= p) return INF;
        if (r - l == 1) return line.get(p);
        return min({line.get(p), ls->query(p), rs->query(p)});
    }
};

```

4.3 Persistent SegmentTree

```

struct Seg {
    Seg *ls{}, *rs{};
    int l, r;
    i64 sum{};
    Seg(Seg* p) { (*this) = *p; }
    Seg(int _l, int _r, const vector<int> &v) : l{_l}, r{
        _r} {
        if (r - l == 1) {
            sum = v[l];

```

```

        return;
    }
    int mid = 1 + r >> 1;
    ls = new Seg(1, mid, v);
    rs = new Seg(mid, r, v);
    pull();
}
void pull() {
    sum = ls->sum + rs->sum;
}
Seg* modify(int p, int v) {
    Seg* ret = new Seg(this);
    if (r - l == 1) {
        ret->sum = v;
        return ret;
    }
    if (p < (1 + r >> 1)) ret->ls = ret->ls->modify(p, v);
    else ret->rs = ret->rs->modify(p, v);
    ret->pull();
    return ret;
}
i64 query(int x, int y) {
    if (y <= 1 or r <= x) return 0;
    if (x <= 1 and r <= y) return sum;
    return ls->query(x, y) + rs->query(x, y);
}
};

```

4.4 Blackmagic

```

#include <bits/extc++.h>
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
#include <ext/pb_ds/hash_policy.hpp>
#include <ext/pb_ds/priority_queue.hpp>
using namespace __gnu_pbds;
template<class T>
using BST = tree<T, null_type, less<T>, rb_tree_tag,
    tree_order_statistics_node_update>;
gnu_pbds::priority_queue<node, decltype(cmp),
    pairing_heap_tag> pq(cmp);
gp_hash_table<int, gnu_pbds::priority_queue<node>::
    point_iterator> pqPos;
bst.insert((x << 20) + i);
bst.erase(bst.lower_bound(x << 20));
bst.order_of_key(x << 20) + 1;
*bst.find_by_order(x - 1) >> 20;
*--bst.lower_bound(x << 20) >> 20;
*bst.upper_bound((x + 1) << 20) >> 20;

```

4.5 Centroid Decomposition

```

struct CenDec {
    vector<vector<pair<int, int>>> anc;
    vector<int> Mdis;
    CenDec(const vector<vector<int>> &G) : anc(G.size()),
        Mdis(G.size(), INF) {
        const int n = G.size();
        vector<int> siz(n);
        vector<bool> vis(n);
        function<int(int, int)> getsiz = [&](int u, int f)
        {
            siz[u] = 1;
            for (int v : G[u]) if (v != f and !vis[v])
                siz[u] += getsiz(v, u);
            return siz[u];
        };
        function<int(int, int, int)> find = [&](int u, int
            f, int s) {
            for (int v : G[u]) if (v != f and !vis[v])
                if (siz[v] * 2 >= s) return find(v, u, s);
            return u;
        };
        function<void(int, int, int, int)> caldis = [&](int
            u, int f, int a, int d) {
            anc[u].emplace_back(a, d);
            for (int v : G[u]) if (v != f and !vis[v])
                caldis(v, u, a, d + 1);
        };
        function<void(int)> build = [&](int u) {
            u = find(u, u, getsiz(u, u));
            vis[u] = 1;
            for (int v : G[u]) if (!vis[v]) {

```

```

        caldis(v, u, u, 1);
        build(v);
    }
    vis[u] = 0;
};
build(0);
}
void add(int p) {
    Mdis[p] = 0;
    for (auto [v, d] : anc[p])
        chmin(Mdis[v], d);
}
int que(int p) {
    int r = Mdis[p];
    for (auto [v, d] : anc[p])
        chmin(r, Mdis[v] + d);
    return r;
}
};

```

5 Math

5.1 Theorem

- Pick's theorem

$$A = i + \frac{b}{2} - 1$$

- Laplacian matrix

$$L = D - A$$

- Extended Catalan number

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

- Derangement $D_n = (n-1)(D_{n-1} + D_{n-2})$

- Möbius

$$\sum_{i|n} \mu(i) = [n=1] \sum_{i|n} \phi(i) = n$$

- Inversion formula

$$f(n) = \sum_{i=0}^n \binom{n}{i} g(i) \quad g(n) = \sum_{i=0}^n (-1)^{n-i} \binom{n}{i} f(i)$$

$$f(n) = \sum_{d|n} g(d) \quad g(n) = \sum_{d|n} \mu\left(\frac{n}{d}\right) f(d)$$

- Sum of powers

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

$$\sum_{j=0}^m \binom{m+1}{j} B_j^- = 0$$

$$\text{note: } B_1^+ = -B_1^- \quad B_i^+ = B_i^-$$

- Cipolla's algorithm

$$\left(\frac{u}{p}\right) = u^{\frac{p-1}{2}}$$

$$1. \left(\frac{a^2 - n}{p}\right) = -1$$

$$2. x = (a + \sqrt{a^2 - n})^{\frac{p+1}{2}}$$

- High order residue

$$[d^{\frac{p-1}{n(p-1)}} \equiv 1]$$

- Packing and Covering

$$|MaximumIndependentSet| + |MinimumVertexCover| = |V|$$

- König's theorem

$$|maximummatching| = |minimumvertexcover|$$

- Dilworth's theorem

$$width = |largestantichain| = |smallestchaindecomposition|$$

- Mirsky's theorem

$$height = |longestchain| = |smallestantichaindecomposition| = |minimumanticliquepartition|$$

- Triangle center

$$- G : (1,)$$

$$- O : (a^2(b^2 + c^2 - a^2),) = (\sin 2A,)$$

$$- I : (a,) = (\sin A)$$

$$- E : (-a, b, c) = (-\sin A, \sin B, \sin C)$$

$$- H : \left(\frac{1}{b^2 + c^2 - a^2},\right) = (\tan A,)$$

5.2 Exgcd

```

pair<i64, i64> exgcd(i64 a, i64 b) { // ax + by = 1
    if (b == 0) return {1, 0};
    auto [x, y] = exgcd(b, a % b);
    return {y, x - a / b * y};
};

```

5.3 CRT

```

i64 CRT(vector<pair<i64, i64>> E) {
    i128 R = 0, M = 1;
    for (auto [r, m] : E) {
        i128 d = r - R, g = gcd<i64>(M, m);
        if (d % g != 0) return -1;
        i128 x = exgcd(M / g, m / g).ff * d / g;
        R += M * x;
        M = M * m / g;
        R = (R % M + M) % M;
    }
    return R;
}

```

5.4 Factorize

```

struct Factorize {
    i64 fmul(i64 a, i64 b, i64 p) {
        return (i128)a * b % p;
    }
    i64 fpow(i64 a, i64 b, i64 p) {
        i64 res = 1;
        for (; b; b >>= 1, a = fmul(a, a, p))
            if (b & 1) res = fmul(res, a, p);
        return res;
    }
    bool Check(i64 a, i64 u, i64 n, int t) {
        a = fpow(a, u, n);
        if (a == 0 or a == 1 or a == n - 1) return true;
        for (int i = 0; i < t; i++) {
            a = fmul(a, a, n);
            if (a == 1) return false;
            if (a == n - 1) return true;
        }
        return false;
    };
    bool IsPrime(i64 n) {
        constexpr array<i64, 7> kChk{2, 235, 9375, 28178,
            450775, 9780504, 1795265022};
        // for int: {2, 7, 61}
        if (n < 2) return false;
        if (n % 2 == 0) return n == 2;
        i64 u = n - 1;
        int t = 0;
        while (u % 2 == 0) u >>= 1, t++;
        for (auto v : kChk) if (!Check(v, u, n, t)) return
            false;
        return true;
    }
    i64 PollardRho(i64 n) {
        if (n % 2 == 0) return 2;
        i64 x = 2, y = 2, d = 1, p = 1;
        auto f = [](i64 x, i64 n, i64 p) -> i64 {
            return ((i128)x * x % n + p) % n;
        };
        while (true) {
            x = f(x, n, p);
            y = f(f(y, n, p), n, p);
            d = __gcd(abs(x - y), n);
            if (d != n and d != 1) return d;
            if (d == n) ++p;
        }
    }
};

```

5.5 NTT

```

// 17 -> 3
// 97 -> 5
// 193 -> 5
// 998244353 -> 3
// 985661441 -> 3
constexpr i64 cpow(i64 a, i64 b, i64 m) {
    i64 ret = 1;
    for (; b; b >>= 1, a = a * a % m)
        if (b & 1) ret = ret * a % m;
}

```

```

    return ret;
};
template<i64 M, i64 G>
struct NTT {
    static constexpr i64 iG = cpow(G, M - 2, M);
    void operator()(vector<i64> &v, bool inv) {
        int n = v.size();
        for (int i = 0, j = 0; i < n; i++) {
            if (i < j) swap(v[i], v[j]);
            for (int k = n / 2; (j ^= k) < k; k /= 2);
        }
        for (int mid = 1; mid < n; mid *= 2) {
            i64 w = cpow((inv ? iG : G), (M - 1) / (mid + mid), M);
            for (int i = 0; i < n; i += mid * 2) {
                i64 now = 1;
                for (int j = i; j < i + mid; j++, now = now * w % M) {
                    i64 x = v[j], y = v[j + mid];
                    v[j] = (x + y * now) % M;
                    v[j + mid] = (x - y * now) % M;
                }
            }
        }
        if (inv) {
            i64 in = cpow(n, M - 2, M);
            for (int i = 0; i < n; i++) v[i] = v[i] * in % M;
        }
    }
};
template<i64 M, i64 G>
vector<i64> convolution(vector<i64> f, vector<i64> g) {
    NTT<M, G> ntt;
    int sum = f.size() + g.size() - 1;
    int len = bit_ceil((u64)sum);
    f.resize(len); g.resize(len);
    ntt(f, 0), ntt(g, 0);
    for (int i = 0; i < len; i++) (f[i] *= g[i]) %= M;
    ntt(f, 1);
    f.resize(sum);
    for (int i = 0; i < sum; i++) if (f[i] < 0) f[i] += M;
    return f;
}
vector<i64> convolution_ll(const vector<i64> &f, const vector<i64> &g) {
    constexpr i64 M1 = 998244353, G1 = 3;
    constexpr i64 M2 = 985661441, G2 = 3;
    constexpr i64 M1M2 = M1 * M2;
    constexpr i64 M1m1 = M2 * cpow(M2, M1 - 2, M1);
    constexpr i64 M2m2 = M1 * cpow(M1, M2 - 2, M2);
    auto c1 = convolution<M1, G1>(f, g);
    auto c2 = convolution<M2, G2>(f, g);
    for (int i = 0; i < c1.size(); i++) {
        c1[i] = ((i128)c1[i] * M1m1 + (i128)c2[i] * M2m2) % M1M2;
    }
    return c1;
}

```

5.6 FWT

1. XOR Convolution

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

2. OR Convolution

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

3. AND Convolution

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

5.7 FWT

```

void xorfwf(int v[], int l, int r) {
    if (r - l == 1) return;
    int m = l + r >> 1;
    xorfwf(v, l, m), xorfwf(v, m, r);
    for (int i = l, j = m; i < m; ++i, ++j) {
        int x = v[i] + v[j];
        v[j] = v[i] - v[j], v[i] = x;
    }
}

```

```

}

void xorifwt(int v[], int l, int r) {
    if (r - l == 1) return;
    int m = l + r >> 1;
    for (int i = l, j = m; i < m; ++i, ++j) {
        int x = (v[i] + v[j]) / 2;
        v[j] = (v[i] - v[j]) / 2, v[i] = x;
    }
    xorifwt(v, l, m), xorifwt(v, m, r);
}

void andfwf(int v[], int l, int r) {
    if (r - l == 1) return;
    int m = l + r >> 1;
    andfwf(v, l, m), andfwf(v, m, r);
    for (int i = l, j = m; i < m; ++i, ++j) v[i] += v[j];
}

void andifwt(int v[], int l, int r) {
    if (r - l == 1) return;
    int m = l + r >> 1;
    andifwt(v, l, m), andifwt(v, m, r);
    for (int i = l, j = m; i < m; ++i, ++j) v[i] -= v[j];
}

void orfwf(int v[], int l, int r) {
    if (r - l == 1) return;
    int m = l + r >> 1;
    orfwf(v, l, m), orfwf(v, m, r);
    for (int i = l, j = m; i < m; ++i, ++j) v[j] += v[i];
}

void orifwt(int v[], int l, int r) {
    if (r - l == 1) return;
    int m = l + r >> 1;
    orifwt(v, l, m), orifwt(v, m, r);
    for (int i = l, j = m; i < m; ++i, ++j) v[j] -= v[i];
}

```

5.8 Lucas

```

i64 Lucas(i64 N, i64 M, i64 D) { // C(N, M) mod D
    auto Factor = [&](i64 x) -> vector<pair<i64, i64>> {
        vector<pair<i64, i64>> r;
        for (i64 i = 2; x > 1; i++)
            if (x % i == 0) {
                i64 c = 0;
                while (x % i == 0) x /= i, c++;
                r.emplace_back(i, c);
            }
        return r;
    };
    auto Pow = [&](i64 a, i64 b, i64 m) -> i64 {
        i64 r = 1;
        for (; b; b >>= 1, a = a * a % m)
            if (b & 1) r = r * a % m;
        return r;
    };
    vector<pair<i64, i64>> E;
    for (auto [p, q] : Factor(D)) {
        const i64 mod = Pow(p, q, 1 << 30);
        auto CountFact = [&](i64 x) -> i64 {
            i64 c = 0;
            while (x) c += (x /= p);
            return c;
        };
        auto CountBino = [&](i64 x, i64 y) { return CountFact(x) - CountFact(y) - CountFact(x - y); };
        auto Inv = [&](i64 x) -> i64 { return exgcd(x, mod).ff % mod + mod; };
        vector<i64> pre(mod + 1);
        pre[0] = pre[1] = 1;
        for (i64 i = 2; i <= mod; i++) pre[i] = (i % p == 0 ? 1 : i) * pre[i - 1] % mod;
        function<i64(i64)> FactMod = [&](i64 n) -> i64 {
            if (n == 0) return 1;
            return FactMod(n / p) * Pow(pre[mod], n / mod, mod) % mod * pre[n % mod] % mod;
        };
        auto BinoMod = [&](i64 x, i64 y) -> i64 {
            return FactMod(x) * Inv(FactMod(y)) % mod * Inv(FactMod(x - y)) % mod;
        };
    }
}

```



```

};
i64 r = BinoMod(N, M) * Pow(p, CountBino(N, M), mod) % mod;
E.emplace_back(r, mod);
};
return CRT(E);
}

```

5.9 Berlekamp Massey

```

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

```

5.10 Gauss Elimination

```

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

```

5.11 Linear Equation

```

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

```

```

swap(r[p], r[i]), swap(c[z], c[i]);
for (int j = 0; j < n; ++j) {
    if (i == j) continue;
    double z = d[r[j]][c[i]] / d[r[i]][c[i]];
    for (int k = 0; k < m; ++k) d[r[j]][c[k]] -= z * d[r[i]][c[k]];
    aug[r[j]] -= z * aug[r[i]];
}
}
vector<vector<double>> fd(n, vector<double>(m));
vector<double> faug(n), x(n);
for (int i = 0; i < n; ++i) {
    for (int j = 0; j < m; ++j) fd[i][j] = d[r[i]][c[j]];
    faug[i] = aug[r[i]];
}
d = fd, aug = faug;
for (int i = n - 1; i >= 0; --i) {
    double p = 0.0;
    for (int j = i + 1; j < n; ++j) p += d[i][j] * x[j];
    x[i] = (aug[i] - p) / d[i][i];
}
for (int i = 0; i < n; ++i) sol[c[i]] = x[i];
}

```

5.12 LinearRec

```

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

```

5.13 SubsetConv

```

vector<int> SubsetConv(int n, const vector<int> &f, const vector<int> &g) {
    const int m = 1 << n;
    vector<vector<int>> a(n + 1, vector<int>(m)), b(n + 1, vector<int>(m));
    for (int i = 0; i < m; ++i) {
        a[__builtin_popcount(i)][i] = f[i];
        b[__builtin_popcount(i)][i] = g[i];
    }
    for (int i = 0; i <= n; ++i) {
        for (int j = 0; j < n; ++j) {
            for (int s = 0; s < m; ++s) {
                if (s >> j & 1) {
                    a[i][s] += a[i][s ^ (1 << j)];
                    b[i][s] += b[i][s ^ (1 << j)];
                }
            }
        }
    }
    vector<vector<int>> c(n + 1, vector<int>(m));
    for (int s = 0; s < m; ++s) {
        for (int i = 0; i <= n; ++i) {
            for (int j = 0; j <= i; ++j) c[i][s] += a[j][s] * b[i - j][s];
        }
    }
}

```

```

}
}
for (int i = 0; i <= n; ++i) {
    for (int j = 0; j < n; ++j) {
        for (int s = 0; s < m; ++s) {
            if (s >> j & 1) c[i][s] -= c[i][s ^ (1 << j)];
        }
    }
}
vector<int> res(m);
for (int i = 0; i < m; ++i) res[i] = c[
    __builtin_popcount(i)][i];
return res;
}

```

5.14 FloorSum

```

// sigma 0 ~ n-1: (a * i + b) / m
i64 floor_sum(i64 n, i64 m, i64 a, i64 b) {
    u64 ans = 0;
    if (a < 0) {
        u64 a2 = (a % m + m) % m;
        ans -= 1ULL * n * (n - 1) / 2 * ((a2 - a) / m);
        a = a2;
    }
    if (b < 0) {
        u64 b2 = (b % m + m) % m;
        ans -= 1ULL * n * ((b2 - b) / m);
        b = b2;
    }
    while (true) {
        if (a >= m) {
            ans += n * (n - 1) / 2 * (a / m);
            a %= m;
        }
        if (b >= m) {
            ans += n * (b / m);
            b %= m;
        }
        u64 y_max = a * n + b;
        if (y_max < m) break;
        n = y_max / m;
        b = y_max % m;
        swap(m, a);
    }
    return ans;
}

```

6 Geometry

6.1 2D Point

```

using Pt = pair<i64, i64>;
constexpr double eps = 1e-9;
Pt operator+(Pt a, Pt b) { return {a.ff + b.ff, a.ss + b.ss}; }
Pt operator-(Pt a, Pt b) { return {a.ff - b.ff, a.ss - b.ss}; }
Pt operator*(Pt a, double b) { return {a.ff * b, a.ss * b}; }
Pt operator/(Pt a, double b) { return {a.ff / b, a.ss / b}; }
double operator*(Pt a, Pt b) { return a.ff * b.ff + a.ss * b.ss; }
double operator^(Pt a, Pt b) { return a.ff * b.ss - a.ss * b.ff; }
double abs(Pt a) { return sqrt(a * a); }
double cro(Pt a, Pt b, Pt c) { return (b - a) ^ (c - a); }
int sig(double x) { return (x > -eps) - (x < eps); }
Pt Inter(Pt a, Pt b, Pt c, Pt d) {
    double s = cro(c, d, a), t = -cro(c, d, b);
    return (a * t + b * s) / (s + t);
}
struct Line {
    Pt a{}, b{};
    Line() {}
    Line(Pt _a, Pt _b) : a{_a}, b{_b} {}
};
Pt Inter(Line L, Line R) {
    return Inter(L.a, L.b, R.a, R.b);
}

```

6.2 Convex Hull

```

vector<Pt> Hull(vector<Pt> P) {
    sort(all(P));
    P.erase(unique(all(P)), P.end());
    P.insert(P.end(), rall(P));
    vector<Pt> stk;
    for (auto p : P) {
        while (stk.size() >= 2 and \
            cro(++stk.rbegin(), stk.back(), p) <= 0 and \
            (*++stk.rbegin() < stk.back()) == (stk.back() <
                p)) {
            stk.pop_back();
        }
        stk.push_back(p);
    }
    stk.pop_back();
    return stk;
}

```

6.3 Convex Hull trick

```

vector<Pt> Hull(vector<Pt> P) {
    sort(all(P));
    P.erase(unique(all(P)), P.end());
    P.insert(P.end(), rall(P));
    vector<Pt> stk;
    for (auto p : P) {
        while (stk.size() >= 2 and \
            cro(++stk.rbegin(), stk.back(), p) <= 0 and \
            (*++stk.rbegin() < stk.back()) == (stk.back() <
                p)) {
            stk.pop_back();
        }
        stk.push_back(p);
    }
    stk.pop_back();
    return stk;
}

```

6.4 Dynamic Convex Hull

```

template<class T, class Comp = less<T>>
struct DynamicHull {
    set<T, Comp> H;
    DynamicHull() {}
    void insert(T p) {
        if (inside(p)) return;
        auto it = H.insert(p).ff;
        while (it != H.begin() and prev(it) != H.begin() \
            and cross(*prev(it), *prev(it), *it) <= 0) {
            it = H.erase(--it);
        }
        while (it != --H.end() and next(it) != --H.end() \
            and cross(*it, *next(it), *next(it), 2)) <= 0) {
            it = --H.erase(++it);
        }
    }
    bool inside(T p) {
        auto it = H.lower_bound(p);
        if (it == H.end()) return false;
        if (it == H.begin()) return p == *it;
        return cross(*prev(it), p, *it) <= 0;
    }
};

```

6.5 Half Plane Intersection

```

vector<Pt> HPI(vector<Line> P) {
    const int n = P.size();
    sort(all(P), [&](Line L, Line R) -> bool {
        Pt u = L.b - L.a, v = R.b - R.a;
        bool f = Pt(sig(u.ff), sig(u.ss)) < Pt{};
        bool g = Pt(sig(v.ff), sig(v.ss)) < Pt{};
        if (f != g) return f < g;
        return (sig(u ^ v) ? sig(u ^ v) : sig(cro(L.a, R.a,
            R.b))) > 0;
    });
    auto Same = [&](Line L, Line R) {
        Pt u = L.b - L.a, v = R.b - R.a;
        return sig(u ^ v) == 0 and sig(u * v) == 1;
    };
    deque<Pt> inter;
    deque<Line> seg;
}

```



```

for (int i = 0; i < n; i++) if (i == 0 or !Same(P[i - 1], P[i])) {
    while (seg.size() >= 2 and sig(cro(inter.back(), P[i].b, P[i].a)) == 1) {
        seg.pop_back(), inter.pop_back();
    }
    while (seg.size() >= 2 and sig(cro(inter[0], P[i].b, P[i].a)) == 1) {
        seg.pop_front(), inter.pop_front();
    }
    if (!seg.empty()) inter.push_back(Inter(seg.back(), P[i]));
    seg.push_back(P[i]);
}
while (seg.size() >= 2 and sig(cro(inter.back(), seg[0].b, seg[0].a)) == 1) {
    seg.pop_back(), inter.pop_back();
}
inter.push_back(Inter(seg[0], seg.back()));
return vector<Pt>(all(inter));
}

```

6.6 Minimal Enclosing Circle

```

using circle = pair<Pt, double>;
struct MES {
    MES() {}
    bool inside(const circle &c, Pt p) {
        return abs(p - c.ff) <= c.ss + eps;
    };
    circle get_cir(Pt a, Pt b) {
        return circle((a + b) / 2., abs(a - b) / 2.);
    }
    circle get_cir(Pt a, Pt b, Pt c) {
        Pt p = (b - a) / 2.;
        p = Pt(-p.ss, p.ff);
        double t = ((c - a) * (c - b)) / (2 * (p * (c - a)));
        p = ((a + b) / 2.) + (p * t);
        return circle(p, abs(p - a));
    }
    circle get_mes(vector<Pt> P) {
        if (P.empty()) return circle(Pt(0, 0), 0);
        mt19937 rng(random_device{}());
        shuffle(all(P), rng);
        circle C{P[0], 0};
        for (int i = 1; i < P.size(); i++) {
            if (inside(C, P[i])) continue;
            C = get_cir(P[i], P[0]);
            for (int j = 1; j < i; j++) {
                if (inside(C, P[j])) continue;
                C = get_cir(P[i], P[j]);
                for (int k = 0; k < j; k++) {
                    if (inside(C, P[k])) continue;
                    C = get_cir(P[i], P[j], P[k]);
                }
            }
        }
        return C;
    }
};

```

6.7 Minkowski

```

vector<Pt> Minkowski(vector<Pt> P, vector<Pt> Q) {
    auto reorder = [&](auto &R) -> void {
        auto cmp = [&](Pt a, Pt b) -> bool {
            return Pt(a.ss, a.ff) < Pt(b.ss, b.ff);
        };
        rotate(R.begin(), min_element(all(R), cmp), R.end());
        R.push_back(R[0]), R.push_back(R[1]);
    };
    const int n = P.size(), m = Q.size();
    reorder(P), reorder(Q);
    vector<Pt> R;
    for (int i = 0, j = 0, s; i < n or j < m; ) {
        R.push_back(P[i] + Q[j]);
        s = sig((P[i + 1] - P[i]) ^ (Q[j + 1] - Q[j]));
        i += (s >= 0), j += (s <= 0);
    }
    return R;
}

```

7 Stringology

7.1 Z-algorithm

```

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

```

7.2 Manacher

```

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

```

7.3 SuffixArray

```

namespace sfx {
#define fup(a, b) for (int i = a; i < b; i++)
#define fdn(a, b) for (int i = b - 1; i >= a; i--)
constexpr int N = 5e5 + 5;
bool t[N * 2];
int H[N], RA[N], x[N], _p[N];
int SA[N * 2], _s[N * 2], _c[N * 2], _q[N * 2];
void pre(int *sa, int *c, int n, int z) {
    fill_n(sa, n, 0), copy_n(c, z, x);
}
void induce(int *sa, int *c, int *s, bool *t, int n, int z) {
    copy_n(c, z - 1, x + 1);
    fup(0, n) if (sa[i] and !t[sa[i] - 1])
        sa[x[sa[sa[i] - 1]]++] = sa[i] - 1;
    copy_n(c, z, x);
    fdn(0, n) if (sa[i] and t[sa[i] - 1])
        sa[--x[sa[sa[i] - 1]]] = sa[i] - 1;
}
void sais(int *s, int *sa, int *p, int *q, bool *t, int *c, int n, int z) {
    bool uniq = t[n - 1] = true;
    int nn = 0, nmzx = -1, *nsa = sa + n, *ns = s + n, last = -1;
    fill_n(c, z, 0);
    fup(0, n) uniq &= ++c[s[i]] < 2;
    partial_sum(c, c + z, c);
    if (uniq) { fup(0, n) sa[--c[s[i]]] = i; return; }
    fdn(0, n - 1)
        t[i] = (s[i] == s[i + 1] ? t[i + 1] : s[i] < s[i + 1]);
    pre(sa, c, n, z);
    fup(1, n) if (t[i] and !t[i - 1])
        sa[--x[s[i]]] = p[q[i] = nn++] = i;
    induce(sa, c, s, t, n, z);
    fup(0, n) if (sa[i] and t[sa[i]] and !t[sa[i] - 1]) {
        bool neq = last < 0 or !equal(s + sa[i], s + p[q[sa[i] + 1], s + last]);
        ns[q[last = sa[i]]] = nmzx += neq;
    }
    sais(ns, nsa, p + nn, q + n, t + n, c + z, nn, nmzx + 1);
    pre(sa, c, n, z);
    fdn(0, nn) sa[--x[s[p[nsa[i]]]]] = p[nsa[i]];
    induce(sa, c, s, t, n, z);
}

```

```

}
vector<int> build(vector<int> s, int n) {
    copy_n(begin(s), n, _s), _s[n] = 0;
    sais(_s, SA, _p, _q, _t, _c, n + 1, 256);
    vector<int> sa(n);
    fup(0, n) sa[i] = SA[i + 1];
    return sa;
}
vector<int> lcp_array(vector<int> &s, vector<int> &sa)
{
    int n = int(s.size());
    vector<int> rnk(n);
    fup(0, n) rnk[sa[i]] = i;
    vector<int> lcp(n - 1);
    int h = 0;
    fup(0, n) {
        if (h > 0) h--;
        if (rnk[i] == 0) continue;
        int j = sa[rnk[i] - 1];
        for (; j + h < n and i + h < n; h++)
            if (s[j + h] != s[i + h]) break;
        lcp[rnk[i] - 1] = h;
    }
    return lcp;
}
}

```

7.4 PalindromicTree

```

struct PAM {
    struct Node {
        int fail, len, dep;
        array<int, 26> ch;
        Node(int _len) : len{_len}, fail{}, ch{}, dep{} {};
    };
    vector<Node> g;
    vector<int> id;
    int odd, even, lst;
    string S;
    int new_node(int len) {
        g.emplace_back(len);
        return g.size() - 1;
    }
    PAM() : odd(new_node(-1)), even(new_node(0)) {
        lst = g[even].fail = odd;
    }
    int up(int p) {
        while (S.rbegin()[g[p].len + 1] != S.back())
            p = g[p].fail;
        return p;
    }
    int add(char c) {
        S += c;
        lst = up(lst);
        c -= 'a';
        if (!g[lst].ch[c]) g[lst].ch[c] = new_node(g[lst].len + 2);
        int p = g[lst].ch[c];
        g[p].fail = (lst == odd ? even : g[up(g[lst].fail)].ch[c]);
        lst = p;
        g[lst].dep = g[g[lst].fail].dep + 1;
        id.push_back(lst);
        return lst;
    }
    void del() {
        S.pop_back();
        id.pop_back();
        lst = id.empty() ? odd : id.back();
    }
};

```

7.5 SmallestRotation

```

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

```

```

}
int pos = (i < n ? i : j);
return t.substr(pos, n);
}

```

8 Misc

8.1 HilbertCurve

```

long long hilbert(int n, int x, int y) {
    long long res = 0;
    for (int s = n / 2; s; s >= 1) {
        int rx = (x & s) > 0;
        int ry = (y & s) > 0;
        res += s * 111 * s * ((3 * rx) ^ ry);
        if (ry == 0) {
            if (rx == 1) x = s - 1 - x, y = s - 1 - y;
            swap(x, y);
        }
    }
    return res;
}

```

8.2 DLX

```

namespace dlx {
int lt[maxn], rg[maxn], up[maxn], dn[maxn], cl[maxn],
    rw[maxn], bt[maxn], s[maxn], head, sz, ans;
void init(int c) {
    for (int i = 0; i < c; ++i) {
        up[i] = dn[i] = bt[i] = i;
        lt[i] = i == 0 ? c : i - 1;
        rg[i] = i == c - 1 ? c : i + 1;
        s[i] = 0;
    }
    rg[c] = 0, lt[c] = c - 1;
    up[c] = dn[c] = -1;
    head = c, sz = c + 1;
}
void insert(int r, const vector<int> &col) {
    if (col.empty()) return;
    int f = sz;
    for (int i = 0; i < (int)col.size(); ++i) {
        int c = col[i], v = sz++;
        dn[bt[c]] = v;
        up[v] = bt[c], bt[c] = v;
        rg[v] = (i + 1 == (int)col.size() ? f : v + 1);
        rw[v] = r, cl[v] = c;
        ++s[c];
        if (i > 0) lt[v] = v - 1;
    }
    lt[f] = sz - 1;
}
void remove(int c) {
    lt[rg[c]] = lt[c], rg[lt[c]] = rg[c];
    for (int i = dn[c]; i != c; i = dn[i]) {
        for (int j = rg[i]; j != i; j = rg[j])
            up[dn[j]] = up[j], dn[up[j]] = dn[j], --s[cl[j]];
    }
}
void restore(int c) {
    for (int i = up[c]; i != c; i = up[i]) {
        for (int j = lt[i]; j != i; j = lt[j])
            ++s[cl[j]], up[dn[j]] = j, dn[up[j]] = j;
    }
    lt[rg[c]] = c, rg[lt[c]] = c;
}
// Call dlx::make after inserting all rows.
void make(int c) {
    for (int i = 0; i < c; ++i)
        dn[bt[i]] = i, up[i] = bt[i];
}
void dfs(int dep) {
    if (dep >= ans) return;
    if (rg[head] == head) return ans = dep, void();
    if (dn[rg[head]] == rg[head]) return;
    int c = rg[head];
    int w = c;
    for (int x = c; x != head; x = rg[x]) if (s[x] < s[w])
        w = x;
    remove(w);
    for (int i = dn[w]; i != w; i = dn[i]) {
        for (int j = rg[i]; j != i; j = rg[j]) remove(cl[j]);
    }
}
}

```

```
    dfs(dep + 1);
    for (int j = lt[i]; j != i; j = lt[j]) restore(cl[j])
    ;
}
restore(w);
}
int solve() {
    ans = 1e9, dfs(0);
    return ans;
}}
```