

## Contents

1	Basic	1
1.1	vimrc	1
1.2	Pragma	1
2	Data Structure	1
2.1	Black Magic	1
2.2	Lazy Segment Tree	1
2.3	Lichao Tree	1
2.4	Linear Basis	2
2.5	Heavy Light Decomposition	2
2.6	Link Cut Tree	2
3	Graph	3
3.1	Bridge CC	3
3.2	Vertex BCC	3
3.3	Strongly Connected Component	3
3.4	Two SAT	4
3.5	Virtual Tree	4
3.6	Dominator Tree	4
3.7	Dinic	4
3.8	Min Cost Max Flow	5
3.9	Stoer Wagner Algorithm	5
3.10	General Matching	5
3.11	Hopcroft Karp Algorithm	6
3.12	Directed MST	6
3.13	Edge Coloring	7
4	Geometry	7
4.1	Basic	7
4.2	2D Convex Hull	7
5	String	8
5.1	KMP	8
5.2	Z Value	8
5.3	Suffix Array	8
5.4	Booth Algorithm	8
5.5	Manacher Algorithm	8
6	Math	8
6.1	Lemma And Theory	8
6.1.1	Pick's Theorem	8
6.1.2	Euler's Planar Graph Theorem	8
6.1.3	Modular inversion recurrence	8
6.2	Numbers	9
6.2.1	Catalan number	9
6.2.2	Primes	9
6.3	Extgcd	9
6.4	Chinese Remainder Theorem	9
6.5	Linear Sieve	9
6.6	Fast Walsh Transform	9
6.7	Floor Sum	9
6.8	Linear Programming	9
6.9	Miller Rabin	10
6.10	Pollard's Rho	10
6.11	Gauss Elimination	10
6.12	Fast Fourier Transform	10
6.13	Primes NTT	10
6.14	Number Theory Transform	11
7	Misc	11
7.1	Josephus Problem	11

## 1 Basic

### 1.1 vimrc

```
set nu rnu is ls=2 hls ts=4 sw=4 et sts=4 ai bs=2 et sc
acd mouse=a encoding=utf-8
syn on
filetype plugin indent on
colo desert
nnoremap <C-a> ggVG
inoremap {<CR> {<CR>}<Esc>O
nnoremap <F8> :w <bar> !g++ -std=c++17 % -o %:r -O2<CR>
nnoremap <F9> :w <bar> !g++ -std=c++17 % -o %:r -Wall -
Wextra -Wconversion -Wshadow -Wfatal-errors -
fsanitize=undefined,address -g -Dgenshin <CR>
nnoremap <F10> :!./%:r <CR>
```

### 1.2 Pragma

```
#pragma GCC optimize("Ofast,no-stack-protector")
#pragma GCC optimize("no-math-errno,unroll-loops")
#pragma GCC target("sse,sse2,sse3,ssse3,sse4")
#pragma GCC target("popcnt,abm,mmx,avx,tune=native")
```

## 2 Data Structure

### 2.1 Black Magic

```
template<typename T>
using pbds_tree = tree<T, null_type, less<T>,
rb_tree_tag, tree_order_statistics_node_update>;
// find_by_order: Like array accessing, order_of_key
```

### 2.2 Lazy Segment Tree

```
// 0-based, [l, r)
// Remember to call init
struct tag {
    // Construct identity element
    tag() { }
    // apply tag
    tag& operator+=(const tag &b) {
        return *this;
    }
};
struct node {
    // Construct identity element
    node() { }
    // Merge two nodes
    node operator+(const node &b) const {
        node res = node();
        return res;
    }
    // Apply tag to this node
    void operator()(const tag &t) {
    }
};
template<typename N, typename T>
struct lazy_segtree {
    N arr[maxn << 1];
    T tag[maxn];
    int n;
    void init(const vector<N> &a) {
        n = a.size();
        for (int i = 0; i < n; i++)
            arr[i + n] = a[i], tag[i] = T();
        for (int i = n - 1; i; i--)
            arr[i] = arr[i << 1] + arr[i << 1 | 1];
    }
    void upd(int p, T v) {
        if (p < n)
            tag[p] += v;
        arr[p](v);
    }
    void pull(int p) {
        for (p >= 1; p; p >= 1) {
            arr[p] = arr[p << 1] + arr[p << 1 | 1];
            arr[p](tag[p]);
        }
    }
    void push(int p) {
        for (int h = __lg(p); h; h--) {
            int i = p >> h;
            upd(i << 1, tag[i]);
            upd(i << 1 | 1, tag[i]);
            tag[i] = T();
        }
    }
    void edt(int l, int r, T v) {
        int tl = l + n, tr = r + n - 1;
        push(tl); push(tr);
        for (l += n, r += n; l < r; l >= 1, r >= 1) {
            if (l & 1)
                upd(l++, v);
            if (r & 1)
                upd(--r, v);
        }
        pull(tl); pull(tr);
    }
    N que(int l, int r) {
        N resl = N(), resr = N();
        int tl = l + n, tr = r + n - 1;
        push(tl); push(tr);
        for (l += n, r += n; l < r; l >= 1, r >= 1) {
            if (l & 1)
                resl = resl + arr[l++];
            if (r & 1)
                resr = arr[--r] + resr;
        }
    }
};
```

```

    }
    return resl + resr;
}
};

```

## 2.3 Lichao Tree

```

struct lichao { // maxn: range
    struct line {
        ll a, b;
        line(): a(0), b(0) { } // or LINF
        line(ll a, ll b): a(a), b(b) { }
        ll operator()(ll x) { return a * x + b; }
    } arr[maxn << 2];
    void insert(int l, int r, int id, line x) {
        int m = (l + r) >> 1;
        if(arr[id](m) < x(m))
            swap(arr[id], x);
        if(l == r - 1)
            return;
        if(arr[id].a < x.a)
            insert(m, r, id << 1 | 1, x);
        else
            insert(l, m, id << 1, x);
    } // change to > if query min
    void insert(ll a, ll b) { insert(0, maxn, 1, line(a, b)); }
    ll que(int l, int r, int id, int p) {
        if(l == r - 1)
            return arr[id](p);
        int m = (l + r) >> 1;
        if(p < m)
            return max(arr[id](p), que(l, m, id << 1, p));
        return max(arr[id](p), que(m, r, id << 1 | 1, p));
    } // chnage to min if query min
    ll que(int p) { return que(0, maxn, 1, p); }
} tree;

```

## 2.4 Linear Basis

```

template<int BITS>
struct linear_basis {
    array<uint64_t, BITS> basis;
    linear_basis() { basis.fill(0); }
    void insert(uint64_t x) {
        for (int i = BITS - 1; i >= 0; i--) if ((x >> i) & 1) {
            if (basis[i] == 0) {
                basis[i] = x;
                return;
            }
            x ^= basis[i];
        }
    }
    bool valid(uint64_t x) {
        for (int i = BITS - 1; i >= 0; i--)
            if ((x >> i) & 1) x ^= basis[i];
        return x == 0;
    }
    uint64_t operator[](int i) { return basis[i]; }
}; // max xor sum: greedy from high bit
// min xor sum: zero(if possible) or min_element

```

## 2.5 Heavy Light Decomposition

```

/* Requirements:
 * N := the count of nodes
 * edge[N] := the edges of the graph
 * Can be modified:
 * tree := Segment Tree or other data structure
 */
struct heavy_light_decomposition {
    int dep[N], pa[N], hea[N], hev[N], pos[N], t;
    int dfs(int u) {
        int mx = 0, sz = 1;
        hev[u] = -1;
        for(int v : edge[u]) {
            if(v == pa[u])
                continue;
            pa[v] = u;
            dep[v] = dep[u] + 1;
            int c = dfs(v);
            if(c > mx)
                mx = c, hev[u] = v;
        }
    }
};

```

```

        sz += c;
    }
    return sz;
}
void find_head(int u, int h) {
    hea[u] = h;
    pos[u] = t++; // 0-indexed !!!
    if(~hev[u])
        find_head(hev[u], h);
    for(int v : edge[u])
        if(v != pa[u] && v != hev[u])
            find_head(v, v);
}
void init(int rt) {
    dfs(rt, rt);
    find_head(rt, rt);
}
/* It is necessary to edit below for every use */
void edt(int a, int b, int v) {
}
int query(int a, int b) { // query path sum
    int res = 0;
    for(; hea[a] != hea[b]; a = pa[hea[a]]) {
        if(dep[hea[a]] < dep[hea[b]])
            swap(a, b);
        res += tree.que(pos[hea[a]], pos[a] + 1);
    }
    if(dep[a] > dep[b])
        swap(a, b);
    return res + tree.que(pos[a], pos[b] + 1);
}
} hld;

```

## 2.6 Link Cut Tree

```

namespace LCT {
    const int N = 1e5 + 25;
    int pa[N], ch[N][2];
    ll dis[N], prv[N], tag[N];
    vector<pair<int, int>> edge[N];
    vector<pair<ll, ll>> eve;
    inline bool dir(int x) { return ch[pa[x]][1] == x; }
    inline bool is_root(int x) { return ch[pa[x]][0] != x
        && ch[pa[x]][1] != x; }
    inline void rotate(int x) {
        int y = pa[x], z = pa[y], d = dir(x);
        if(!is_root(y))
            ch[z][dir(y)] = x;
        pa[x] = z;
        ch[y][d] = ch[x][!d];
        if(ch[x][!d])
            pa[ch[x][!d]] = y;
        ch[x][!d] = y;
        pa[y] = x;
    }
    inline void push_tag(int x) {
        if(!tag[x])
            return;
        prv[x] = tag[x];
        if(ch[x][0])
            tag[ch[x][0]] = tag[x];
        if(ch[x][1])
            tag[ch[x][1]] = tag[x];
        tag[x] = 0;
    }
    void push(int x) {
        if(!is_root(x))
            push(pa[x]);
        push_tag(x);
    }
    inline void splay(int x) {
        push(x);
        while(!is_root(x)) {
            if(int y = pa[x]; !is_root(y))
                rotate(dir(y) == dir(x) ? y : x);
            rotate(x);
        }
    }
    inline void access(ll t, int x) {
        int lst = 0, tx = x;
        while(x) {
            splay(x);
            tx = pa[x];
            if(tx)
                push_tag(tx);
            pa[x] = tx;
            x = tx;
        }
    }
};

```

```

    if(lst) {
        ch[x][1] = lst;
        eve.push_back({prv[x] + dis[x], t + dis[x]});
    }
    lst = x;
    x = pa[x];
}
splay(tx);
if(ch[tx][0])
    tag[ch[tx][0]] = t;
}
void dfs(int u) {
    prv[u] = -LINF;
    for(const auto &[v, c] : edge[u]) {
        if(v == pa[u])
            continue;
        pa[v] = u;
        ch[u][1] = v;
        dis[v] = dis[u] + c;
        dfs(v);
    }
}
};

```

### 3 Graph

#### 3.1 Bridge CC

```

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

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

```

#### 3.2 Vertex BCC

```

class bicon_cc {
private:
    int n, ecnt;

```

```

    vector<vector<pair<int, int>>> G;
    vector<int> bcc, dfn, low, st;
    vector<bool> ap, ins;
    void dfs(int u, int f) {
        dfn[u] = low[u] = dfn[f] + 1;
        int ch = 0;
        for (auto [v, t]: G[u]) if (v != f) {
            if (!ins[t]) {
                st.push_back(t);
                ins[t] = true;
            }
            if (dfn[v]) {
                low[u] = min(low[u], dfn[v]);
                continue;
            }
            ++ch;
            dfs(v, u);
            low[u] = min(low[u], low[v]);
            if (low[v] >= dfn[u]) {
                ap[u] = true;
                while (true) {
                    int eid = st.back();
                    st.pop_back();
                    bcc[eid] = ecnt;
                    if (eid == t) break;
                }
                ecnt++;
            }
        }
        if (ch == 1 && u == f) ap[u] = false;
    }
public:
    void init(int n_) {
        G.clear(); G.resize(n = n_);
        ecnt = 0; ap.assign(n, false);
        low.assign(n, 0); dfn.assign(n, 0);
    }
    void add_edge(int u, int v) {
        G[u].emplace_back(v, ecnt);
        G[v].emplace_back(u, ecnt++);
    }
    void solve() {
        ins.assign(ecnt, false);
        bcc.resize(ecnt); ecnt = 0;
        for (int i = 0; i < n; ++i)
            if (!dfn[i]) dfs(i, i);
    }
    // The id of bcc of the x-th edge (0-indexed)
    int get_id(int x) { return bcc[x]; }
    // Number of bcc
    int count() { return ecnt; }
    bool is_ap(int x) { return ap[x]; }
}; // 0-indexed

```

#### 3.3 Strongly Connected Component

```

namespace scc {
    vector<int> edge[maxn], redge[maxn];
    stack<int, vector<int>> st;
    bool vis[maxn];
    void dfs(int u) {
        vis[u] = true;
        for(int v : edge[u])
            if(!vis[v])
                dfs(v);
        st.push(u);
    }
    void dfs2(int u, vector<int> &pa) {
        for(int v : redge[u])
            if(!pa[v])
                pa[v] = pa[u], dfs2(v, pa);
    }
    void add_edge(int u, int v) {
        edge[u].push_back(v);
        redge[v].push_back(u);
    }
    // pa[i]: scc id of all nodes in topo order
    vector<int> solve(int n) {
        vector<int> pa(n + 1);
        for(int i = 1; i <= n; i++)
            if(!vis[i])
                dfs(i);
        int id = 1; // start from 1

```

```

while(!st.empty()) {
    int u = st.top();
    st.pop();
    if(!pa[u])
        pa[u] = id++, dfs2(u, pa);
}
return pa;
} // 1-based
};

```

### 3.4 Two SAT

```

// maxn >= 2 * n (n: number of variables)
// clauses: (x, y) = x V y, -x if neg, var are 1-based
// return empty is no solution
vector<bool> solve(int n, const vector<pair<int, int>>
    &clauses) {
    auto id = [&](int x) { return abs(x) + n * (x < 0); };
    for(const auto &a, b : clauses) {
        scc::add_edge(id(-a), id(b));
        scc::add_edge(id(-b), id(a));
    }
    auto pa = scc::solve(n * 2);
    vector<bool> ans(n + 1);
    for(int i = 1; i <= n; i++) {
        if(pa[i] == pa[i + n])
            return vector<bool>();
        ans[i] = pa[i] > pa[i + n];
    }
    return ans;
}

```

### 3.5 Virtual Tree

```

// dfn: the dfs order, vs: important points, r: root
vector<pair<int, int>> build(vector<int> vs, int r) {
    vector<pair<int, int>> res;
    sort(vs.begin(), vs.end(), [](int i, int j) {
        return dfn[i] < dfn[j]; });
    vector<int> s = {r};
    for (int v : vs) if (v != r) {
        if (int o = lca(v, s.back()); o != s.back()) {
            while (s.size() >= 2) {
                if (dfn[s[s.size() - 2]] < dfn[o]) break;
                res.emplace_back(s[s.size() - 2], s.back());
                s.pop_back();
            }
            if (s.back() != o) {
                res.emplace_back(o, s.back());
                s.back() = o;
            }
        }
        s.push_back(v);
    }
    for (size_t i = 1; i < s.size(); ++i)
        res.emplace_back(s[i - 1], s[i]);
    return res; // (x, y): x->y
} // The returned virtual tree contains r (root).

```

### 3.6 Dominator Tree

```

/* Find dominator tree with root s in O(n)
 * Return the father of each node, *-2 for unreachable
 ** */
struct dominator_tree { // 0-based
    int tk;
    vector<vector<int>> g, r, rdom;
    vector<int> dfn, rev, fa, sdom, dom, val, rp;
    dominator_tree(int n): tk(0), g(n), r(n), rdom(n),
        dfn(n, -1), rev(n, -1), fa(n, -1), sdom(n, -1),
        dom(n, -1), val(n, -1), rp(n, -1) {}
    void add_edge(int x, int y) { g[x].push_back(y); }
    void dfs(int x) {
        rev[dfn[x] = tk] = x;
        fa[tk] = sdom[tk] = val[tk] = tk;
        tk++;
        for (int u : g[x]) {
            if (dfn[u] == -1) dfs(u), rp[dfn[u]] = dfn[x];
            r[dfn[u]].push_back(dfn[x]);
        }
    }
    void merge(int x, int y) { fa[x] = y; }
    int find(int x, int c = 0) {

```

```

        if (fa[x] == x) return c ? -1 : x;
        if (int p = find(fa[x], 1); p != -1) {
            if (sdom[val[x]] > sdom[val[fa[x]]])
                val[x] = val[fa[x]];
            fa[x] = p;
            return c ? p : val[x];
        } else {
            return c ? fa[x] : val[x];
        }
    }
}
vector<int> build(int s, int n) {
    dfs(s);
    for (int i = tk - 1; i >= 0; --i) {
        for (int u : r[i])
            sdom[i] = min(sdom[i], sdom[find(u)]);
        if (i) rdom[sdom[i]].push_back(i);
        for (int u : rdom[i]) {
            int p = find(u);
            dom[u] = (sdom[p] == i ? i : p);
        }
        if (i) merge(i, rp[i]);
    }
    vector<int> p(n, -2);
    p[s] = -1;
    for (int i = 1; i < tk; ++i)
        if (sdom[i] != dom[i]) dom[i] = dom[dom[i]];
    for (int i = 1; i < tk; ++i)
        p[rev[i]] = rev[dom[i]];
    return p;
}
};

```

### 3.7 Dinic

```

// Return max flor from s to t. INF, LINF and maxn
// required
template<typename T> // maxn: edge/node counts
struct dinic { // T: int or LL, up to range of flow
    const T IN_INF = (is_same_v<T, int>) ? INF : LINF;
    struct E {
        int v; T c; int r;
        E(int v, T c, int r):
            v(v), c(c), r(r){}
    };
    vector<E> adj[maxn];
    pair<int, int> is[maxn]; // counts of edges
    void add_edge(int u, int v, T c, int i = 0) {
        is[i] = {u, adj[u].size()};
        adj[u].push_back(E(v, c, (int) adj[v].size()));
        adj[v].push_back(E(u, 0, (int) adj[u].size() - 1));
    }
    int n, s, t;
    void init(int nn, int ss, int tt) {
        n = nn, s = ss, t = tt;
        for (int i = 0; i <= n; ++i)
            adj[i].clear();
    }
    int le[maxn], it[maxn];
    int bfs() {
        fill(le, le + maxn, -1); le[s] = 0;
        queue<int> q; q.push(s);
        while (!q.empty()) {
            int u = q.front(); q.pop();
            for (auto [v, c, r] : adj[u]) {
                if (c > 0 && le[v] == -1)
                    le[v] = le[u] + 1, q.push(v);
            }
        }
        return ~le[t];
    }
    T dfs(int u, T f) {
        if (u == t) return f;
        for (int &i = it[u]; i < (int) adj[u].size(); ++i) {
            auto &[v, c, r] = adj[u][i];
            if (c > 0 && le[v] == le[u] + 1) {
                T d = dfs(v, min(c, f));
                if (d > 0) {
                    c -= d;
                    adj[v][r].c += d;
                    return d;
                }
            }
        }
    }
}

```

```

    }
    return 0;
}
T flow() {
    T ans = 0, d;
    while (bfs()) {
        fill(it, it + maxn, 0);
        while ((d = dfs(s, IN_INF)) > 0) ans += d;
    }
    return ans;
}
T rest(int i) {
    return adj[is[i].first][is[i].second].c;
}
};

```

### 3.8 Min Cost Max Flow

```

struct cost_flow { // maxn: node count
    static const int64_t INF = 102938475610293847LL;
    struct Edge {
        int v, r;
        int64_t f, c;
        Edge(int a, int b, int _c, int d):v(a),r(b),f(_c),c(d) {}
    };
    int n, s, t, prv[maxn], prvl[maxn], inq[maxn];
    int64_t dis[maxn], fl, cost;
    vector<Edge> E[maxn];
    void init(int _n, int _s, int _t) {
        n = _n; s = _s; t = _t;
        for (int i = 0; i < n; i++) E[i].clear();
        fl = cost = 0;
    }
    void add_edge(int u, int v, int64_t f, int64_t c) {
        E[u].push_back(Edge(v, E[v].size(), f, c));
        E[v].push_back(Edge(u, E[u].size()-1, 0, -c));
    }
    pair<int64_t, int64_t> flow() {
        while (true) {
            for (int i = 0; i < n; i++) {
                dis[i] = INF;
                inq[i] = 0;
            }
            dis[s] = 0;
            queue<int> que;
            que.push(s);
            while (!que.empty()) {
                int u = que.front(); que.pop();
                inq[u] = 0;
                for (int i = 0; i < E[u].size(); i++) {
                    int v = E[u][i].v;
                    int64_t w = E[u][i].c;
                    if (E[u][i].f > 0 && dis[v] > dis[u] + w) {
                        prv[v] = u; prvl[v] = i;
                        dis[v] = dis[u] + w;
                        if (!inq[v]) {
                            inq[v] = 1;
                            que.push(v);
                        }
                    }
                }
            }
            if (dis[t] == INF) break;
            int64_t tf = INF;
            for (int v = t, u, l; v != s; v = u) {
                u = prv[v]; l = prvl[v];
                tf = min(tf, E[u][l].f);
            }
            for (int v = t, u, l; v != s; v = u) {
                u = prv[v]; l = prvl[v];
                E[u][l].f -= tf;
                E[v][E[u][l].r].f += tf;
            }
            cost += tf * dis[t];
            fl += tf;
        }
        return {fl, cost};
    }
};

```

### 3.9 Stoer Wagner Algorithm

```

// return global min cut in O(n^3)
struct SW { // 1-based
    int edge[maxn][maxn], wei[maxn], n;
    bool vis[maxn], del[maxn];
    void init(int _n) {
        n = _n; MEM(edge, 0); MEM(del, 0);
    }
    void add_edge(int u, int v, int w) {
        edge[u][v] += w; edge[v][u] += w;
    }
    void search(int &s, int &t) {
        MEM(wei, 0); MEM(vis, 0);
        s = t = -1;
        while (true) {
            int mx = -1;
            for (int i = 1; i <= n; i++) {
                if (del[i] || vis[i]) continue;
                if (mx == -1 || wei[mx] < wei[i])
                    mx = i;
            }
            if (mx == -1) break;
            vis[mx] = true;
            s = t; t = mx;
            for (int i = 1; i <= n; i++)
                if (!vis[i] && !del[i])
                    wei[i] += edge[mx][i];
        }
    }
    int solve() {
        int ret = INF;
        for (int i = 1; i < n; i++) {
            int x, y;
            search(x, y);
            ret = min(ret, wei[y]);
            del[y] = true;
            for (int j = 1; j <= n; j++) {
                edge[x][j] += edge[y][j];
                edge[j][x] += edge[y][j];
            }
        }
        return ret;
    }
} sw;

```

### 3.10 General Matching

```

// Find max matching on general graph in O(|V|^3)
vector<int> max_matching(vector<vector<int>>> g) {
    int n = g.size();
    vector<int> match(n+1, n), pre(n+1, n), que;
    vector<int> s(n+1), mark(n+1), pa(n+1);
    function<int(int)> fnd = [&](int x) {
        if (x == pa[x]) return x;
        return pa[x] = fnd(pa[x]);
    };
    auto lca = [&](int x, int y) {
        static int tk = 0;
        tk++;
        x = fnd(x);
        y = fnd(y);
        for (; swap(x, y))
            if (x != n) {
                if (mark[x] == tk)
                    return x;
                mark[x] = tk;
                x = fnd(pre[match[x]]);
            }
    };
    auto blossom = [&](int x, int y, int l) {
        while (fnd(x) != l) {
            pre[x] = y;
            y = match[x];
            if (s[y] == 1)
                que.push_back(y), s[y] = 0;
            if (pa[x] == x) pa[x] = 1;
            if (pa[y] == y) pa[y] = 1;
            x = pre[y];
        }
    };
    auto bfs = [&](int r) {
        fill(s.begin(), s.end(), -1);
        iota(pa.begin(), pa.end(), 0);
        que = {r}; s[r] = 0;
    };
}

```

```

for(int it = 0; it < que.size(); it++) {
    int x = que[it];
    for(int u : g[x]) {
        if(s[u] == -1) {
            pre[u] = x;
            s[u] = 1;
            if(match[u] == n) {
                for(int a = u, b = x, lst;
                    b != n; a = lst, b = pre[a]) {
                    lst = match[b];
                    match[b] = a;
                    match[a] = b;
                }
                return;
            }
            que.push_back(match[u]);
            s[match[u]] = 0;
        }
        else if(s[u] == 0 && fnd(u) != fnd(x)) {
            int l = lca(u, x);
            blossom(x, u, l);
            blossom(u, x, l);
        }
    }
}
};
for(int i = 0; i < n; i++)
    if(match[i] == n) bfs(i);
match.resize(n);
for(int i = 0; i < n; i++)
    if(match[i] == n) match[i] = -1;
return match;
} // 0-based

```

### 3.11 Hopcroft Karp Algorithm

```

// Find maximum bipartite matching in  $O(E\sqrt{V})$ 
// g: edges for all nodes at left side
vector<int> hopcroft_karp(vector<vector<int>> g, int l,
    int r) {
    vector<int> match_l(l, -1), match_r(r, -1);
    vector<int> dis(l);
    vector<bool> vis(l);
    while(true) {
        queue<int> que;
        for(int i = 0; i < l; i++) {
            if(match_l[i] == -1)
                dis[i] = 0, que.push(i);
            else
                dis[i] = -1;
            vis[i] = false;
        }
        while(!que.empty()) {
            int x = que.front();
            que.pop();
            for(int y : g[x])
                if(match_r[y] != -1 && dis[match_r[y]] == -1) {
                    dis[match_r[y]] = dis[x] + 1;
                    que.push(match_r[y]);
                }
        }
        auto dfs = [&](auto dfs, int x) {
            vis[x] = true;
            for(int y : g[x]) {
                if(match_r[y] == -1) {
                    match_l[x] = y;
                    match_r[y] = x;
                    return true;
                }
                else if(dis[match_r[y]] == dis[x] + 1
                    && !vis[match_r[y]]) {
                    && dfs(dfs, match_r[y]);
                    match_l[x] = y;
                    match_r[y] = x;
                    return true;
                }
            }
            return false;
        };
        bool ok = true;
        for(int i = 0; i < l; i++)
            if(match_l[i] == -1 && dfs(dfs, i))
                ok = false;
    }
}

```

```

if(ok)
    break;
}
return match_l;
} // 0-based

```

### 3.12 Directed MST

```

// Find minimum directed minimum spanning tree in  $O(E \log V)$ 
// DSU rollback is required
// Return parent of all nodes, -1 for unreachable ones
// and root
struct dmst_edge { int a, b; ll w; };
struct dmst_node { // lazy skew heap node
    dmst_edge key;
    dmst_node *l, *r;
    ll delta;
    void prop() {
        key.w += delta;
        if (l) l->delta += delta;
        if (r) r->delta += delta;
        delta = 0;
    }
    dmst_edge top() { prop(); return key; }
};
dmst_node *dmst_merge(dmst_node *a, dmst_node *b) {
    if (!a || !b) return a ? b;
    a->prop();
    b->prop();
    if (a->key.w > b->key.w) swap(a, b);
    swap(a->l, (a->r = dmst_merge(b, a->r)));
    return a;
}
void dmst_pop(dmst_node*& a) {
    a->prop();
    a = dmst_merge(a->l, a->r);
}
pair<ll, vector<int>> dmst(int n, int r, const vector<
    dmst_edge*& g) {
    dsu_undo uf(n);
    vector<dmst_node*> heap(n);
    vector<dmst_node*> tmp;
    for (dmst_edge e : g) {
        tmp.push_back(new dmst_node {e});
        heap[e.b] = dmst_merge(heap[e.b], tmp.back());
    }
    ll res = 0;
    vector<int> seen(n, -1), path(n), par(n);
    seen[r] = r;
    vector<dmst_edge> Q(n), in(n, {-1, -1}), comp;
    deque<tuple<int, int, vector<dmst_edge>>> cys;
    for (int s = 0; s < n; s++) {
        int u = s, qi = 0, w;
        while (seen[u] < 0) {
            if (!heap[u]) return {-1, {}};
            dmst_edge e = heap[u]->top();
            heap[u]->delta -= e.w;
            dmst_pop(heap[u]);
            Q[qi] = e;
            path[qi++] = u;
            seen[u] = s;
            res += e.w;
            u = uf.find(e.a);
            if (seen[u] == s) { // found cycle, contract
                dmst_node* cyc = 0;
                int end = qi, time = uf.time();
                do {
                    cyc = dmst_merge(cyc, heap[w = path[--qi]]);
                } while (uf.join(u, w));
                u = uf.find(u);
                heap[u] = cyc;
                seen[u] = -1;
                cys.push_front({u, time, {Q[qi], Q[end]}});
            }
        }
        for (int i = 0; i < qi; i++)
            in[uf.find(Q[i].b)] = Q[i];
    }
    for (auto& [u, t, comp] : cys) { // restore sol (
        optional)
        uf.rollback(t);
    }
}

```



```

    indmst_edge indmst_edge = in[u];
    for (auto& e : comp) in[uf.find(e.b)] = e;
    in[uf.find(indmst_edge.b)] = indmst_edge;
}
for (int i = 0; i < n; i++)
    par[i] = in[i].a;
for (auto &a : tmp)
    delete a;
return {res, par};
}

```

### 3.13 Edge Coloring

*/\* Find a edge coloring using at most d+1 colors, where d is the max deg, in  $O(V^3)$   
 \* mat[i][j] is the color between i, j in 1-based (0 for no edge)  
 \* use recolor() to add edge. Calculation is done in every recolor \*/*

```

struct edge_coloring { // 0-based
    int n;
    int mat[maxn][maxn];
    bool vis[maxn], col[maxn];
    void init(int _n) { n = _n; } // remember to init
    int check_conflict(int x, int loc) {
        for (int i = 0; i < n; i++)
            if (mat[x][i] == loc)
                return i;
        return n;
    }
    int get_block(int x) {
        memset(col, 0, sizeof col);
        for (int i = 0; i < n; i++) col[mat[x][i]] = 1;
        for (int i = 1; i < n; i++) if (!col[i]) return i;
        return n;
    }
    void recolor(int x, int y) {
        int pre_mat = get_block(y);
        int conflict = check_conflict(x, pre_mat);
        memset(vis, 0, sizeof vis);
        vis[y] = 1;
        vector<pair<int, int>> mat_line;
        mat_line.push_back({y, pre_mat});
        while (conflict != n && !vis[conflict]) {
            vis[conflict] = 1;
            y = conflict;
            pre_mat = get_block(y);
            mat_line.push_back({y, pre_mat});
            conflict = check_conflict(x, pre_mat);
        }
        if (conflict == n) {
            for (auto t : mat_line) {
                mat[x][t.first] = t.second;
                mat[t.first][x] = t.second;
            }
        }
        else {
            int pre_mat_x = get_block(x);
            int conflict_x = check_conflict(conflict, pre_mat_x);
            mat[x][conflict] = pre_mat_x;
            mat[conflict][x] = pre_mat_x;
            while (conflict_x != n) {
                int tmp = check_conflict(conflict_x, pre_mat);
                mat[conflict][conflict_x] = pre_mat;
                mat[conflict_x][conflict] = pre_mat;
                conflict = conflict_x;
                conflict_x = tmp;
                swap(pre_mat_x, pre_mat);
            }
            recolor(x, mat_line[0].first);
        }
    }
} mg;

```

## 4 Geometry

### 4.1 Basic

```

struct point {
    ld x, y;
    point() { }
    point(ld a, ld b): x(a), y(b) { }
}

```

```

point operator-(const point &b) const {
    return point(x - b.x, y - b.y);
}
point operator+(const point &b) const {
    return point(x + b.x, y + b.y);
}
point operator*(ld r) const {
    return point(x * r, y * r);
}
point operator/(ld r) const {
    return point(x / r, y / r);
}
bool operator<(const point &b) const {
    return x == b.x ? y < b.y : x < b.x; }
ld dis2() { return x * x + y * y; }
ld dis() { return sqrt(dis2()); }
point perp() { return point(-y, x); }
point norm() {
    ld d = dis();
    return point(x / d, y / d);
}
ld cross(const point &a, const point &b, const point &c) {
    auto x = b - a, y = c - a;
    return x.x * y.y - y.x * x.y;
}
ld dot(const point &a, const point &b, const point &c) {
    auto x = b - a, y = c - a;
    return x.x * y.x + x.y * y.y;
}
ld area(const point &a, const point &b, const point &c) {
    return ld(cross(a, b, c)) / 2;
}
static inline bool eq(ld a, ld b) { return abs(a - b) < EPS; }
int sgn(ld v) {
    return v > 0 ? 1 : (v < 0 ? -1 : 0);
}
int ori(point a, point b, point c) {
    return sgn(cross(a, b, c));
}
bool collinearity(point a, point b, point c) {
    return ori(a, b, c) == 0;
}
bool btw(point p, point a, point b) {
    return collinearity(p, a, b) && sgn(dot(p, a, b)) <= 0;
}
point projection(point p1, point p2, point p3) {
    return (p2 - p1) * dot(p1, p2, p3) / (p2 - p1).dis2();
}
using Line = pair<point, point>;
bool seg_intersect(Line a, Line b) {
    point p1, p2, p3, p4;
    tie(p1, p2) = a;
    tie(p3, p4) = b;
    if (btw(p1, p3, p4) || btw(p2, p3, p4) || btw(p3, p1, p2) || btw(p4, p1, p2))
        return true;
    return ori(p1, p2, p3) * ori(p1, p2, p4) < 0 && ori(p3, p4, p1) * ori(p3, p4, p2) < 0;
}
point intersect(Line a, Line b) {
    point p1, p2, p3, p4;
    tie(p1, p2) = a;
    tie(p3, p4) = b;
    ld a123 = cross(p1, p2, p3);
    ld a124 = cross(p1, p2, p4);
    return (p4 * a123 - p3 * a124) / (a123 - a124);
}

```

### 4.2 2D Convex Hull

*// returns a convex hull in counterclockwise order  
 // for a non-strict one, change cross >= to >*

```

vector<point> convex_hull(vector<point> p) {
    sort(p.begin(), p.end());
    if (p[0] == p.back()) return {p[0]};
    int n = p.size(), t = 0;

```

```

vector<point> h(n + 1);
for (int _ = 2, s = 0; _--; s = --t, reverse(p.begin(), p.end()))
    for (point i : p) {
        while (t > s + 1 && cross(i, h[t-1], h[t-2]) >= 0)
            t--;
        h[t++] = i;
    }
return h.resize(t), h;
} // not tested, but trust ckiseki!

```

## 5 String

### 5.1 KMP

```

vector<int> kmp(const string &s) {
    int n = s.size();
    vector<int> dp(n);
    for (int i = 1, j = 0; i < n; i++) {
        while (j && s[i] != s[j])
            j = dp[j - 1];
        if (s[i] == s[j])
            j++;
        dp[i] = j;
    }
    return dp;
}

```

### 5.2 Z Value

```

// Return Z value of string s in O(|S|)
// Note that z[0] = |S|
vector<int> Zalgo(const string &s) {
    vector<int> z(s.size(), (int) s.size());
    for (int i = 1, l = 0, r = 0; i < z[0]; ++i) {
        int j = clamp(r - i, 0, z[i - 1]);
        while (i + j < z[0] && s[i + j] == s[j])
            j++;
        if (i + (z[i] = j) > r)
            r = i + z[i];
    }
    return z;
}

```

### 5.3 Suffix Array

```

int sa[maxn], tmp[2][maxn], c[maxn];
void get_sa(const string &s) { // m: char set
    int *x = tmp[0], *y = tmp[1], m = 256, n = s.size();
    for (int i = 0; i < m; i++) c[i] = 0;
    for (int i = 0; i < n; i++) c[x[i]] = s[i] + 1;
    for (int i = 1; i < m; i++) c[i] += c[i - 1];
    for (int i = n - 1; i >= 0; --i) sa[--c[x[i]]] = i;
    for (int k = 1; k < n; k <= 1) {
        for (int i = 0; i < m; i++) c[i] = 0;
        for (int i = 0; i < n; i++) c[x[i]] += 1;
        for (int i = 1; i < m; i++) c[i] += c[i - 1];
        int p = 0;
        for (int i = n - k; i < n; i++) y[p++] = i;
        for (int i = 0; i < n; i++)
            if (sa[i] >= k) y[p++] = sa[i] - k;
        for (int i = n - 1; i >= 0; --i) sa[--c[x[y[i]]]] = y[i];
        y[sa[0]] = p = 0;
        for (int i = 1; i < n; i++) {
            int a = sa[i], b = sa[i - 1];
            if (x[a] == x[b] && a + k < n && b + k < n && x[a + k] == x[b + k]) { }
            else p++;
            y[sa[i]] = p;
        }
        if (n == p + 1)
            break;
        swap(x, y);
        m = p + 1;
    }
} // sa[i]: index which ranks i
int rk[maxn], lcp[maxn]; // lcp[i] : lcp with i-1
void get_lcp(const string &s) {
    int n = s.size(), val = 0;
    for (int i = 0; i < n; i++) rk[sa[i]] = i;
    for (int i = 0; i < n; i++) {
        if (rk[i] == 0) lcp[rk[i]] = 0;
        else {

```

```

            if (val) val--;
            int p = sa[rk[i] - 1];
            while (val + i < n && val + p < n && s[val + i] == s[val + p])
                val++;
            lcp[rk[i]] = val;
        }
    }
}

```

### 5.4 Booth Algorithm

```

// return start index of minimum rotation in O(|s|)
int min_rotation(string s) {
    s += s;
    int k = 0;
    vector<int> f(s.size(), -1);
    for (int j = 1; j < s.size(); j++) {
        int i = f[j - k - 1];
        for (i = f[j - k - 1];
             i != -1 && s[j] != s[i + k + 1]; i = f[i])
            if (s[k + i + 1] > s[j])
                k = j - i - 1;
        if (i == -1 && s[j] != s[k + i + 1]) {
            if (s[j] < s[k + i + 1])
                k = j;
            f[j - k] = -1;
        }
        else
            f[j - k] = i + 1;
    }
    return k;
}

```

### 5.5 Manacher Algorithm

```

vector<int> manacher_algorithm(string s) {
    int n = 2 * s.size() + 1;
    string t(n, 0);
    vector<int> len(n); // len[i]: max length when mid at i
    for (int i = 0; i < n; i++) {
        if (i & 1)
            t[i] = s[i / 2];
    }
    for (int i = 0, l = 0, r = -1; i < n; i++) {
        len[i] = (i <= r ? min(len[2 * l - i], r - i) : 0);
        while (i - len[i] >= 0 && i + len[i] < n && t[i - len[i]] == t[i + len[i]])
            len[i]++;
        len[i]--;
        if (i + len[i] > r)
            l = i, r = i + len[i];
    }
    return len;
}

```

## 6 Math

### 6.1 Lemma And Theory

#### 6.1.1 Pick's Theorem

For a simple polygon, its area  $A$  can be written as  $A = i + \frac{b}{2} - 1$  in which  $i$  is the number of points that are strictly interior to the polygon and  $b$  is the number of points that are on the polygon's boundary.

#### 6.1.2 Euler's Planar Graph Theorem

$F$ : number of regions bounded by edges.  
 $V - E + F = C + 1, E \leq 3V - 6$

#### 6.1.3 Modular inversion recurrence

For some prime  $p$ ,

$$inv_i = \begin{cases} 1 & i = 1 \\ p - \lfloor \frac{p}{i} \rfloor \times inv_{(p \bmod i)} & 1 < i < p \end{cases}$$



## 6.2 Numbers

### 6.2.1 Catalan number

Start from  $n = 0 : 1, 1, 2, 5, 14, 42, 132, 429, 1430, 4862, 16796, 58786, \dots$

$$C_n = \frac{1}{n+1} \binom{2n}{n} = \frac{(2n)!}{(n+1)!n!} = \prod_{k=2}^n \frac{n+k}{k}$$

$$C_n = \binom{2n}{n} - \binom{2n}{n+1}$$

Recurrence

$$C_0 = 1$$

$$C_{n+1} = \sum_{i=0}^n C_i C_{n-i}$$

$$C_{n+1} = \frac{2(2n+1)}{n+2} C_n$$

### 6.2.2 Primes

12721, 13331, 14341, 75577

999997771, 999991231, 1000000007, 1000000009, 1000696969

$$10^{12} + 39, 10^{15} + 37$$

## 6.3 Extgcd

```
// return (d, x, y) s.t. ax+by=d=gcd(a,b)
template<typename T>
tuple<T, T, T> extgcd(T a, T b) {
    if(!b) return make_tuple(a, 1, 0);
    auto [d, x, y] = extgcd(b, a % b);
    return make_tuple(d, y, x - (a / b) * y);
}
```

## 6.4 Chinese Remainder Theorem

```
// x % m1 = x1, x % m2 = x2
ll chre(ll x1, ll m1, ll x2, ll m2){
    ll g = __gcd(m1, m2);
    if ((x2 - x1) % g) return -1; // no solution
    m1 /= g; m2 /= g;
    ll p = get<1>(extgcd(m1, m2));
    ll lcm = m1 * m2 * g;
    ll res = p * (x2 - x1) * m1 + x1;
    // might overflow for above two lines, be cautious
    return (res % lcm + lcm) % lcm;
}
```

## 6.5 Linear Sieve

```
int least_prime_divisor[maxn];
vector<int> pr;
void linear_sieve() {
    for(int i = 2; i < maxn; i++) {
        if(!least_prime_divisor[i]) {
            pr.push_back(i);
            least_prime_divisor[i] = i;
        }
        for(int p : pr) {
            if(1LL * i * p >= maxn) break;
            least_prime_divisor[i * p] = p;
            if(i % p == 0) break;
        }
    }
}
```

## 6.6 Fast Walsh Transform

```
/* do not move ta,tb, default for xor
 * remove last 2 lines for non-xor
 * or convolution:
 * x[i]=ta,x[j]=ta+tb; x[i]=ta,x[j]=tb-ta for inv
 * and convolution:
 * x[i]=ta+tb,x[j]=tb; x[i]=ta-tb,x[j]=tb for inv */
void fwt(int x[], int N, bool inv = false) {
    for(int d = 1; d < N; d <= 1) {
        for(int s = 0, d2 = d * 2; s < N; s += d2)
            for(int i = s, j = s + d; i < s + d; i++, j++) {
                int ta = x[i], tb = x[j];
                x[i] = modadd(ta, tb);
                x[j] = modsub(ta, tb);
            }
    }
}
```

```
}
if(inv) for(int i = 0, invn = modinv(N); i < N; i++)
    x[i] = modmul(x[i], invn);
} // N: array Len
```

## 6.7 Floor Sum

```
// @param n `n < 2^32`
// @param m `1 <= m < 2^32`
// @return sum_{i=0}^{n-1} floor((ai + b)/m) mod 2^64
ull floor_sum_unsigned(ull n, ull m, ull a, ull b) {
    ull ans = 0;
    while (true) {
        if (a >= m) {
            ans += n * (n - 1) / 2 * (a / m); a %= m;
        }
        if (b >= m) {
            ans += n * (b / m); b %= m;
        }
        ull y_max = a * n + b;
        if (y_max < m) break;
        // y_max < m * (n + 1)
        // floor(y_max / m) <= n
        n = (ull)(y_max / m), b = (ull)(y_max % m);
        swap(m, a);
    }
    return ans;
}
ll floor_sum(ll n, ll m, ll a, ll b) {
    ull ans = 0;
    if (a < 0) {
        ull a2 = (a % m + m) % m;
        ans -= 1ULL * n * (n - 1) / 2 * ((a2 - a) / m);
        a = a2;
    }
    if (b < 0) {
        ull b2 = (b % m + m) % m;
        ans -= 1ULL * n * ((b2 - b) / m);
        b = b2;
    }
    return ans + floor_sum_unsigned(n, m, a, b);
}
```

## 6.8 Linear Programming

```
/* M constraints, i-th constraint is:
 \sum_{j=0}^{n-1} A[i][j] * x_j <= B[i]
 Let v = \sum_{j=0}^{n-1} C[j] * x_j
 maximize v satisfying constraints
 sol[i] = x_i
 remind the precision error */
struct Simplex { // 0-based
    using T = long double;
    static const int N = 410, M = 30010;
    const T eps = 1e-7;
    int n, m;
    int Left[M], Down[N];
    T a[M][N], b[M], c[N], v, sol[N];
    bool eq (T a, T b) { return fabs(a - b) < eps; }
    bool ls (T a, T b) { return a < b && !eq(a, b); }
    void init(int _n, int _m) {
        n = _n, m = _m, v = 0;
        for (int i = 0; i < m; ++i) for (int j = 0; j < n; ++j) {
            a[i][j] = 0;
        }
        for (int i = 0; i < m; ++i) b[i] = 0;
        for (int i = 0; i < n; ++i) c[i] = sol[i] = 0;
    }
    void pivot (int x, int y) {
        swap(Left[x], Down[y]);
        T k = a[x][y]; a[x][y] = 1;
        vector<int> nz;
        for (int i = 0; i < n; ++i) {
            a[x][i] /= k;
            if(!eq(a[x][i], 0)) nz.push_back(i);
        }
        b[x] /= k;
        for (int i = 0; i < m; ++i) {
            if (i == x || eq(a[i][y], 0)) continue;
            k = a[i][y], a[i][y] = 0;
            b[i] -= k * b[x];
            for (int j : nz) a[i][j] -= k * a[x][j];
        }
    }
}
```

```

    if (eq(c[y], 0)) return;
    k = c[y], c[y] = 0, v += k * b[x];
    for (int i : nz) c[i] -= k * a[x][i];
}
// 0: found solution, 1: no feasible solution, 2:
// unbounded
int solve() {
    for (int i = 0; i < n; ++i) Down[i] = i;
    for (int i = 0; i < m; ++i) Left[i] = n + i;
    while (1) {
        int x = -1, y = -1;
        for (int i = 0; i < m; ++i) if (ls(b[i], 0) && (x == -1 || b[i] < b[x])) x = i;
        if (x == -1) break;
        for (int i = 0; i < n; ++i) if (ls(a[x][i], 0) && (y == -1 || a[x][i] < a[x][y])) y = i;
        if (y == -1) return 1;
        pivot(x, y);
    }
    while (1) {
        int x = -1, y = -1;
        for (int i = 0; i < n; ++i) if (ls(0, c[i]) && (y == -1 || c[i] > c[y])) y = i;
        if (y == -1) break;
        for (int i = 0; i < m; ++i) if (ls(0, a[i][y]) && (x == -1 || b[i] / a[i][y] < b[x] / a[x][y])) x = i;
        if (x == -1) return 2;
        pivot(x, y);
    }
    for (int i = 0; i < m; ++i) if (Left[i] < n) sol[Left[i]] = b[i];
    return 0;
}
} LP;

```

## 6.9 Miller Rabin

```

bool is_prime(ull x) { // need modular pow(mpow)
    static auto witn = [](ull a, ull u, ull n, int t) {
        if (!a) return false;
        while (t--) {
            ull a2 = __uint128_t(a) * a % n;
            if (a2 == 1 && a != 1 && a != n - 1) return true;
            a = a2;
        }
        return a != 1;
    };
    if (x < 2) return false;
    if (!(x & 1)) return x == 2;
    int t = __builtin_ctzll(x - 1);
    ull odd = (x - 1) >> t;
    for (ull m: {2, 325, 9375, 28178, 450775, 9780504, 1795265022}) {
        if (witn(mpow(m % x, odd, x), odd, x, t))
            return false;
    }
    return true;
}

```

## 6.10 Pollard's Rho

```

ull f(ull x, ull k, ull m) {
    return (__uint128_t(x) * x + k) % m;
}
// does not work when n is prime
// return any non-trivial factor
ull pollard_rho(ull n) {
    if (!(n & 1)) return 2;
    mt19937 rnd(120821011);
    while (true) {
        ull y = 2, yy = y, x = rnd() % n, t = 1;
        for (ull sz = 2; t == 1; sz <= 1, y = yy) {
            for (ull i = 0; t == 1 && i < sz; ++i) {
                yy = f(yy, x, n);
                t = __gcd(yy > y ? yy - y : y - yy, n);
            }
        }
        if (t != 1 && t != n) return t;
    }
}

```

## 6.11 Gauss Elimination

```

void gauss_elimination(vector<vector<double>> &d) {
    int n = d.size(), m = d[0].size();
    for (int i = 0; i < m; ++i) {
        int p = -1;
        for (int j = i; j < n; ++j) {
            if (fabs(d[j][i]) < eps) continue;
            if (p == -1 || fabs(d[j][i]) > fabs(d[p][i])) p = j;
        }
        if (p == -1) continue;
        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];
        }
    }
} // Not tested

```

## 6.12 Fast Fourier Transform

```

using cplx = complex<double>;
const double pi = acos(-1);
cplx omega[maxn * 4];
void prefft(int n) {
    for (int i = 0; i <= n; i++)
        omega[i] = cplx(cos(2 * pi * i / n), sin(2 * pi * i / n));
}
void fft(vector<cplx> &v, int n) {
    int z = __builtin_ctz(n) - 1;
    for (int i = 0; i < n; i++) {
        int x = 0, j = 0;
        for (; (1 << j) < n; j++) x ^= (i >> j & 1) << (z - j);
        if (x > i) swap(v[x], v[i]);
    }
    for (int s = 2; s <= n; s <= 1) {
        int z = s >> 1;
        for (int i = 0; i < n; i += s) {
            for (int k = 0; k < z; k++) {
                cplx x = v[i + z + k] * omega[n / s * k];
                v[i + z + k] = v[i + k] - x;
                v[i + k] = v[i + k] + x;
            }
        }
    }
}
void ifft(vector<cplx> &v, int n) {
    fft(v, n); reverse(v.begin() + 1, v.end());
    for (int i = 0; i < n; i++) v[i] = v[i] * cplx(1.0 / n, 0);
}
v1 convolution(const v1 &a, const v1 &b) {
    // Should be able to handle N <= 10^5, C <= 10^4
    int sz = 1, tot = a.size() + b.size() - 1;
    while (sz < tot) sz <= 1;
    prefft(sz);
    vector<cplx> v(sz);
    for (int i = 0; i < sz; i++) {
        double re = i < a.size() ? a[i] : 0;
        double im = i < b.size() ? b[i] : 0;
        v[i] = cplx(re, im);
    }
    fft(v, sz);
    for (int i = 0; i <= sz / 2; i++) {
        int j = (sz - i) & (sz - 1);
        cplx x = (v[i] + conj(v[j])) * (v[i] - conj(v[j])) * cplx(0, -0.25);
        if (j != i) v[j] = (v[j] + conj(v[i])) * (v[j] - conj(v[i])) * cplx(0, -0.25);
        v[i] = x;
    }
    ifft(v, sz);
    v1 c(sz);
    for (int i = 0; i < sz; i++) c[i] = round(v[i].real());
    c.resize(tot);
    return c;
}

```

## 6.13 3 Primes NTT

```

// MOD: arbitrary prime

```

```

const int M1 = 998244353;
const int M2 = 1004535809;
const int M3 = 2013265921;
int super_big_crt(int64_t A, int64_t B, int64_t C) {
    static_assert(M1 <= M2 && M2 <= M3);
    ll r12 = mpow(M1, M2 - 2, M2);
    ll r13 = mpow(M1, M3 - 2, M3);
    ll r23 = mpow(M2, M3 - 2, M3);
    ll M1M2 = 1LL * M1 * M2 % MOD;
    B = (B - A + M2) * r12 % M2;
    C = (C - A + M3) * r13 % M3;
    C = (C - B + M3) * r23 % M3;
    return (A + B * M1 + C * M1M2) % MOD;
} // return ans % MOD
return k;
} // both not tested

```

## 6.14 Number Theory Transform

```

/* mod | g | maxn possible values:
998244353 | 3 | 8388608
1004535809 | 3 | 2097152
2013265921 | 31 | 134217728 */
template <int mod, int G, int maxn>
struct NTT {
    ll mpow(ll a, ll b) {
        ll res = 1;
        for(; b; b >>= 1, a = a * a % mod)
            if(b & 1)
                res = res * a % mod;
        return res;
    }
    static_assert(maxn == (maxn & -maxn));
    int roots[maxn];
    NTT() {
        ll r = mpow(G, (mod - 1) / maxn);
        for(int i = maxn >> 1; i; i >>= 1) {
            roots[i] = 1;
            for(int j = 1; j < i; j++)
                roots[i + j] = roots[i + j - 1] * r % mod;
            r = r * r % mod;
        }
    }
    // n must be 2^k, and 0 <= f[i] < mod
    // n >= the size after convolution
    void operator()(vector<ll> &f, int n, bool inv =
        false) {
        for(int i = 0, j = 0; i < n; i++) {
            if(i < j) swap(f[i], f[j]);
            for(int k = n >> 1; (j ^ k) < k; k >>= 1);
        }
        for(int s = 1; s < n; s *= 2) {
            for(int i = 0; i < n; i += s * 2) {
                for(int j = 0; j < s; j++) {
                    ll a = f[i + j];
                    ll b = f[i + j + s] * roots[s + j] % mod;
                    f[i + j] = (a + b) % mod;
                    f[i + j + s] = (a - b + mod) % mod;
                }
            }
        }
        if(inv) {
            int invn = mpow(n, mod - 2);
            for(int i = 0; i < n; i++)
                f[i] = f[i] * invn % mod;
            reverse(f.begin() + 1, f.end());
        }
    }
};

```

## 7 Misc

### 7.1 Josephus Problem

```

// n people kill m for each turn
int f(int n, int m) {
    int s = 0;
    for (int i = 2; i <= n; i++)
        s = (s + m) % i;
    return s;
}
// died at kth
int kth(int n, int m, int k){
    if (m == 1) return n-1;
    for (k = k*m+m-1; k >= n; k = k-n+(k-n)/(m-1));
}

```