Team Reference Document

Heltion

December 3, 2021

Contents

1 Misc

1.1 Makefile

```
%:%.cpp
	g++ $< -o $@ -std=c++17 -O2 -Wall -Wextra
```

1.2 Template

```cpp
#pragma GCC optimize("O3")
#pragma GCC target("avx2")
#include<bits/stdc++.h>
using namespace std;
using LL = long long;
int main(){
    cin.tie(nullptr)->sync_with_stdio(false);
    return 0;
}
```

2 Data Struture

2.1 Treap

```cpp
struct Node{
    int v, p, size;
    Node *L, *R;
    Node(int v, int p, Node* L = nullptr, Node* R = nullptr) : v(v), p(p), size(1), L(L), R(R) {
    }
    Node* copy(Node* L, Node* R) {
        this->L = L;
        this->R = R;
        return this;
    }
    //persistent
    Node* copy(Node* L, Node* R) {
        return new Node(v, p, L, R);
    }
};
namespace Treap {
    mt19937 rng;
    int size(Node* p) {
        return p ? p->size : 0;
    }
    Node* update(Node* p) {
        p->size = 1 + size(p->L) + size(p->R);
        return p;
    }
    pair<Node*, Node*> split(Node* p, int v) {
        if (not p) return {};
        if (p->v <= v) {
            auto [L, R] = split(p->R, v);
            return {update(p->copy(p->L, L)), R};
        }
        auto [L, R] = split(p->L, v);
        return {L, update(p->copy(R, p->R))};
    }
    Node* merge(Node*L, Node* R) {
        if (not L) return R;
        if (not R) return L;
        if (L->p < R->p) return update(L->copy(L->L, merge(L->R, R)));
        return update(R->copy(merge(L, R->L), R->R));
    }
    Node* insert(Node* p, int v) {
        auto [L, R] = split(p, v);
        return merge(merge(L, new Node(v, rng())), R);
    }
    Node* remove(Node* p, int v) {
        auto [LM, R] = split(p, v);
        auto [L, M] = split(LM, v - 1);
        return merge(merge(L, M ? merge(M->L, M->R) : M), R);
    }
    int rank(Node* p, int v) {
        if (not p) return 1;
        if (p->v >= v) return rank(p->L, v);
        return 1 + size(p->L) + rank(p->R, v);
    }
    Node* kth(Node* p, int k) {
        if (not p or k > p->size) return nullptr;
        if (k <= size(p->L)) return kth(p->L, k);
        if (k == size(p->L) + 1) return p;
        return kth(p->R, k - size(p->L) - 1);
```

```
    }
    Node* pedecessor(Node* p, int v) {
        if (not p) return p;
        if (p->v >= v) return pedecessor(p->L, v);
        auto R = pedecessor(p->R, v);
        return R ? R : p;
    }
    Node* successor(Node* p, int v) {
        if (not p) return p;
        if (p->v <= v) return successor(p->R, v);
        auto L = successor(p->L, v);
        return L ? L : p;
    }
};
```

2.2 Cartesian Tree

```
vector<int> cartesianTree(const vector<int>& v) {
    int n = v.size();
    vector<int> p(n, -1);
    stack<int> s;
    for (int i = 0, t = -1; i < n; i += 1) {
        for(t = -1; not s.empty() and v[s.top()] > v[i]; s.pop()) t = s.top();
        if (not s.empty()) p[i] = s.top();
        if (t != -1) p[t] = i;
        s.push(i);
    }
    return p;
}
```

2.3 Convex Hull(Line Add Get Max)

```
struct line{
    static bool Q;
    mutable LL k, m, p;
    bool operator<(const line& o)const{
        return Q ? p < o.p : k < o.k;
    }
};
bool line::Q = false;
struct lines : multiset<line> {
    //(for doubles, use inf = 1/.0, div(a,b) = a/b)
    const LL inf = LLONG_MAX;
    LL div(LL a, LL b){
        return a / b - ((a ^ b) < 0 and a % b);
    }
    bool isect(iterator x, iterator y) {
        if(y == end()) return x->p = inf, false;
        if(x->k == y->k) x->p = x->m > y->m ? inf : -inf;
        else x->p = div(y->m - x->m, x->k - y->k);
        return x->p >= y->p;
    }
    void add(LL k, LL m) {
        line::Q = false;
        auto z = insert({k, m, 0}), y = z++, x = y;
        while(isect(y, z)) z = erase(z);
        if(x != begin() and isect(--x, y)) isect(x, y = erase(y));
        while((y = x) != begin() and (--x)->p >= y->p)
        isect(x, erase(y));
    }
    LL query(LL x) {
        line::Q = true;
        auto l = lower_bound({0, 0, x});
        return l->k * x + l->m;
    }
};
```

2.4 Link Cut Tree(Subtree Add Subtree Sum)

```
struct node{
    node* p;
    array<node*, 2> c;
    int rev, size, vsize;
    LL v, sum, vsum, add, vadd;
    node(){
        p = c[0] = c[1] = nullptr;
        size = 1;
        v = vsize = sum = vsum = add = vadd = rev = 0;
    }
    int d(){
        if(p){
```

```
            if(p->c[0] == this) return 0;
            if(p->c[1] == this) return 1;
        }
        return -1;
    }
    void flush(){
        if(~d()) p->flush();
        down();
    }
    void rotate(){
        int pd = d();
        auto pp = p, pc = c[not pd];
        p = pp->p; if(~pp->d()) pp->p->c[pp->d()] = this;
        pp->c[pd] = pc; if(pc) pc->p = pp;
        c[not pd] = pp; pp->p = this;
        pp->up();
    }
    void splay(){
        for(flush(); ~d(); rotate())
            if(~p->d()) (d() == p->d() ? p : this)->rotate();
        up();
    }
    void reverse(){
        rev ^= 1;
        swap(c[0], c[1]);
    }
    void addf(LL x){
        v += x;
        sum += size * x;
        vsum += vsize * x;
        add += x;
        vadd += x;
    }
    void change(node* c, int v){
        if(not v) c->addf(vadd);
        vsize += v ? c->size : -c->size;
        vsum += v ? c->sum : -c->sum;
        if(v) c->addf(-vadd);
    }
    void up(){
        size = 1 + vsize;
        sum = v + vsum;
        for(auto x : c) if(x){
            size += x->size;
            sum += x->sum;
        }
    }
    void down(){
        if(rev) for(auto x : c) if(x) x->reverse();
        if(add) for(auto x : c) if(x) x->addf(add);
        rev = add = 0;
    }
};
struct link_cut_tree : vector<node>{
    link_cut_tree(int n) : vector<node>(n){}
    node* get(int u){
        return data() + u;
    }
    void access(int u){
        for(node *v = get(u), *p = nullptr; v; p = v, v = v->p){
            v->splay();
            if(v->c[1]) v->change(v->c[1], 1);
            if(p) v->change(p, 0);
            v->c[1] = p;
            v->up();
        }
        get(u)->splay();
    }
    void make_root(int u){
        access(u);
        at(u).reverse();
    }
    void link(int u, int v){
        access(u);
        make_root(v);
        at(v).p = get(u);
        at(u).c[1] = get(v);
        at(u).up();
    }
    void cut(int u, int v){
        make_root(u);
        access(v);
        at(v).c[0]->p = nullptr;
        at(v).c[0] = nullptr;
        at(v).up();
    }
};
```

2.5 Segment Beats!

```
#define tm ((tl + tr) >> 1)
struct Node{
    static constexpr LL inf = __LONG_LONG_MAX__;
    Node *ls, *rs;
    LL mx, smx, cmx, mn, smn, cmn, sm, tmx, tmn, tadd;
    Node() {
        tmx = -inf;
        tmn = inf;
        tadd = 0;
    }
    void init(LL x) {
        mx = mn = sm = x;
        cmx = cmn = 1;
        smx = tmx = -inf;
        smn = tmn = inf;
        tadd = 0;
    }
    void push_up() {
        sm = ls->sm + rs->sm;
        mx = max(ls->mx, rs->mx);
        smx = max(ls->mx == mx ? ls->smx : ls->mx, rs->mx == mx ? rs->smx : rs->mx);
        cmx = (ls->mx == mx ? ls->cmx : 0) + (rs->mx == mx ? rs->cmx : 0);
        mn = min(ls->mn, rs->mn);
        smn = min(ls->mn == mn ? ls->smn : ls->mn, rs->mn == mn ? rs->smn : rs->mn);
        cmn = (ls->mn == mn ? ls->cmn : 0) + (rs->mn == mn ? rs->cmn : 0);
    }
    void add(int tl, int tr, LL x) {
        sm += (tr - tl) * x;
        mx += x;
        if (smx != -inf) smx += x;
        if (tmx != -inf) tmx += x;
        mn += x;
        if (smn != inf) smn += x;
        if (tmn != inf) tmn += x;
        tadd += x;
    }
    void chmin(LL x) {
        if (mx <= x) return;
        sm += (x - mx) * cmx;
        if (smn == mx) smn = x;
        if (mn == mx) mn = x;
        if (tmx > x) tmx = x;
        mx = tmn = x;
    }
    void chmax(LL x) {
        if (mn >= x) return;
        sm += (x - mn) * cmn;
        if (smx == mn) smx = x;
        if (mx == mn) mx = x;
        if (tmn < x) tmn = x;
        mn = tmx = x;
    }
    void push_down(int tl, int tr) {
        if (tadd) {
            ls->add(tl, tm, tadd);
            rs->add(tm, tr, tadd);
            tadd = 0;
        }
        if (tmn != inf) {
            ls->chmin(tmn);
            rs->chmin(tmn);
            tmn = inf;
        }
        if (tmx != -inf) {
            ls->chmax(tmx);
            rs->chmax(tmx);
            tmx = -inf;
        }
    }
    void chmin(int tl, int tr, int L, int R, LL x) {
        if (mx <= x) return;
        if (tl >= L and tr <= R and smx < x) return chmin(x);
        push_down(tl, tr);
        if (L < tm) ls->chmin(tl, tm, L, R, x);
        if (R > tm) rs->chmin(tm, tr, L, R, x);
        push_up();
    }
    void chmax(int tl, int tr, int L, int R, LL x) {
        if (mn >= x) return;
        if (tl >= L and tr <= R and smn > x) return chmax(x);
        push_down(tl, tr);
        if (L < tm) ls->chmax(tl, tm, L, R, x);
        if (R > tm) rs->chmax(tm, tr, L, R, x);
        push_up();
    }
    void add(int tl, int tr, int L, int R, LL x) {
        if (tl >= L and tr <= R) return add(tl, tr, x);
        push_down(tl, tr);
        if (L < tm) ls->add(tl, tm, L, R, x);
```

```
        if (R > tm) rs->add(tm, tr, L, R, x);
        push_up();
    }
    LL sum(int tl, int tr, int L, int R) {
        if (tl >= L and tr <= R) return sm;
        push_down(tl, tr);
        LL res = 0;
        if (L < tm) res += ls->sum(tl, tm, L, R);
        if (R > tm) res += rs->sum(tm, tr, L, R);
        return res;
    }
};
struct Segment_Tree_Beats : vector<Node>{
    Node* root;
    int N;
    Segment_Tree_Beats(int N) : vector<Node>(2 * N - 1), N(N){}
    void init(vector<LL>& a) {
        int p = 0;
        function<void(Node*&, int, int)> build = [&](Node*& v, int tl, int tr) {
            v = &at(p ++);
            if (tl + 1 == tr) return v->init(a[tm]);
            build(v->ls, tl, tm);
            build(v->rs, tm, tr);
            v->push_up();
        };
        build(root, 0, N);
    }
    void chmin(int l, int r, LL b) {
        root->chmin(0, N, l, r, b);
    }
    void chmax(int l, int r, LL b) {
        root->chmax(0, N, l, r, b);
    }
    void add(int l, int r, LL b) {
        root->add(0, N, l, r, b);
    }
    LL sum(int l, int r) {
        return root->sum(0, N, l, r);
    }
};
```

3 Graph

3.1 Euler Circuit

```
vector<int> undirected_circuit(int n, const vector<pair<int, int>>& edges) {
    int m = edges.size();
    vector<int> vis(m), res;
    vector<vector<int>> G(n);
    for (int i = 0; i < m; i += 1) {
        G[edges[i].first].push_back(i + 1);
        G[edges[i].second].push_back(-i - 1);
    }
    for (int i = 0; i < n; i += 1) if (G[i].size() & 1) return {};
    vector<vector<int>::const_iterator> it(n);
    for (int i = 0; i < n; i += 1) it[i] = G[i].begin();
    function<void(int)> dfs = [&](int u) {
        for (auto& nxt = it[u]; nxt != G[u].end(); ) {
            int i = abs(*nxt) - 1;
            if (not vis[i]) {
                vis[i] = 1;
                int w = *nxt;
                dfs(*nxt >= 0 ? edges[i].second : edges[i].first);
                res.push_back(-w);
            }
            else nxt = next(nxt);
        }
    };
    for (int i = 0; i < n; i += 1) if (not G[i].empty()) {
        dfs(i);
        break;
    }
    if (res.size() < m) return {};
    return res;
}

vector<int> directed_circuit(int n, const vector<pair<int, int>>& edges) {
    int m = edges.size();
    vector<int> d(n), vis(m), res;
    vector<vector<int>> G(n);
    for (int i = 0; i < m; i += 1) {
        G[edges[i].first].push_back(i);
        d[edges[i].second] += 1;
    }
    for (int i = 0; i < n; i += 1) if (G[i].size() != d[i]) return {};
    vector<vector<int>::const_iterator> it(n);
    for (int i = 0; i < n; i += 1) it[i] = G[i].begin();
```

```
    function<void(int)> dfs = [&](int u) {
        for (auto& nxt = it[u]; nxt != G[u].end(); ) {
            if (not vis[*nxt]) {
                vis[*nxt] = 1;
                int w = *nxt;
                dfs(edges[w].second);
                res.push_back(w);
            }
            else nxt = next(nxt);
        }
    };
    for (int i = 0; i < n; i += 1) if (not G[i].empty()) {
        dfs(i);
        break;
    }
    if (res.size() < m) return {};
    reverse(res.begin(), res.end());
    return res;
}
```

3.2 Strongly Connected Component(Kosaraju)

```
vector<vector<int>> kosaraju(int n, const vector<pair<int, int>>& edges){
    vector<int> vis(n), p;
    vector<vector<int>> res, G(n), Gt(n);
    for (auto [a, b] : edges) {
        G[a].push_back(b);
        Gt[b].push_back(a);
    }
    function<void(int)> dfs = [&](int u){
        vis[u] = 1;
        for(int v : G[u]) if(not vis[v]) dfs(v);
        p.push_back(u);
    };
    function<void(int)> dfst = [&](int u){
        vis[u] = 0;
        res.back().push_back(u);
        for(int v : Gt[u]) if(vis[v]) dfst(v);
    };
    for(int i = 0; i < n; i += 1) if(not vis[i]) dfs(i);
    reverse(p.begin(), p.end());
    for(int i : p) if(vis[i]) {
        res.emplace_back();
        dfst(i);
    }
    return res;
}
```

3.3 K-shortest Paths

```
struct Edge{
    int u, v, w;
};
struct Node{
    int v, h;
    LL w;
    Node *ls, *rs;
    Node(int v, LL w) : v(v), w(w) {
        h = 1;
        ls = rs = nullptr;
    }
};
Node* merge(Node* u, Node* v) {
    if (u == nullptr) return v;
    if (v == nullptr) return u;
    if (u->w > v->w) swap(u, v);
    Node* p = new Node(*u);
    p->rs = merge(u->rs, v);
    if (p->rs != nullptr and (p->ls == nullptr or p->ls->h < p->rs->h)) swap(p->ls, p->rs);
    p->h = (p->rs ? p->rs->h : 0) + 1;
    return p;
}
vector<LL> k_shortest_walk(int N, const vector<Edge>& edges, int S, int T, int K) {
    vector<vector<int>> G(N);
    for (int i = 0; i < (int)edges.size(); i += 1) G[edges[i].v].push_back(i);
    priority_queue<pair<LL, int>, vector<pair<LL, int>>, greater<pair<LL, int>>> pq;
    vector<LL> d(N, -1);
    vector<int> done(N), par(N, -1), p;
    d[T] = 0;
    pq.push({0, T});
    while (not pq.empty()) {
        int u = pq.top().second;
        pq.pop();
        if (done[u]) continue;
        p.push_back(u);
```

```
        done[u] = 1;
        for (int i : G[u]) {
            auto [v, _, w] = edges[i];
            if (d[v] == -1 or d[v] > d[u] + w) {
                d[v] = d[u] + w;
                par[v] = i;
                pq.push({d[v], v});
            }
        }
    }
    if (d[S] == -1) return vector<LL>(K, -1);
    vector<Node*> heap(N);
    for (int i = 0; i < (int)edges.size(); i += 1) {
        auto [u, v, w] = edges[i];
        if (~d[u] and ~d[v] and par[u] != i) heap[u] = merge(heap[u], new Node(v, d[v] + w - d[u]));
    }
    for (int u : p) if (u != T)
        heap[u] = merge(heap[u], heap[edges[par[u]].v]);
    priority_queue<pair<LL, Node*>, vector<pair<LL, Node*>>, greater<pair<LL, Node*>>> q;
    if (heap[S]) q.push({d[S] + heap[S]->w, heap[S]});
    vector<LL> res = {d[S]};
    for (int i = 1; i < K and not q.empty(); i += 1) {
        auto [w, node] = q.top();
        q.pop();
        res.push_back(w);
        if (heap[node->v]) q.push({w + heap[node->v]->w, heap[node->v]});
        for (auto s : {node->ls, node->rs})
            if (s) q.push({w + s->w - node->w, s});
    }
    res.resize(K, -1);
    return res;
}
```

3.4 Articulation Points

```
vector<int> articulation_points(int n, const vector<vector<int>>& G){
    int ts = 0;
    vector<int> dfn(n), low(n), cut(n);
    function<void(int, int)> dfs = [&](int u, int p){
        dfn[u] = low[u] = ts ++;
        int c = 0;
        for(int v : G[u])
            if(not ~dfn[v]){
                dfs(v, u);
                low[u] = min(low[u], low[v]);
                if(low[v] >= dfn[u] and ~p) cut[u] = 1;
                c += 1;
            }
            else low[u] = min(low[u], dfn[v]);
        if(not ~p and c > 1) cut[u] = 1;
    };
    for(int i = 0; i < n; i += 1) if(not dfn[i]) dfs(i, -1);
    vector<int> res;
    for(int i = 0; i < n; i += 1) if(cut[i]) res.push_back(i);
    return res;
}
```

3.5 Dominator Tree(Lengauer-Tarjan)

```
vector<int> dominator_tree(const vector<vector<int>>& G, int r){
    int n = G.size(), ts = 0;
    vector<int> dfn(n, -1), par(n), ord(n), f(n), mn(n), sdom(n), dom(n);
    vector<vector<int>> H(n), I(n);
    for(int i = 0; i < n; i += 1){
        for(int j : G[i]) H[j].push_back(i);
        f[i] = sdom[i] = mn[i] = i;
    }
    function<void(int)> dfs = [&](int u){
        ord[dfn[u] = ts ++] = u;
        for(int v : G[u]) if(not ~dfn[v]){
            par[v] = u;
            dfs(v);
        }
    };
    function<int(int)> find = [&](int u){
        if(u == f[u]) return u;
        int fu = find(f[u]);
        if(dfn[sdom[mn[f[u]]]] < dfn[sdom[mn[u]]]) mn[u] = mn[f[u]];
        return f[u] = fu;
    };
    dfs(r);
    for(int i = n - 1; i; i -= 1){
        int u = ord[i];
        for(int v : H[u]) if(~dfn[v]){
            find(v);
```

```
            if(dfn[sdom[mn[v]]] < dfn[sdom[u]]) sdom[u] = sdom[mn[v]];
        }
        I[sdom[u]].push_back(u);
        u = f[u] = par[u];
        for(int v : I[u]){
            find(v);
            dom[v] = u == sdom[mn[v]] ? u : mn[v];
        }
        I[u].clear();
    }
    for(int u : ord) if(dom[u] != sdom[u]) dom[u] = dom[dom[u]];
    return dom;
}
```

3.6 Directed Minimum Spanning Tree

```
struct Edge{
    int u, v, w;
};
struct node{
    Edge key;
    node *L, *R;
    LL delta;
    node(Edge key) : key(key){
        L = R = nullptr;
        delta = 0;
    }
    void down(){
        key.w += delta;
        if(L) L->delta += delta;
        if(R) R->delta += delta;
        delta = 0;
    }
    Edge top(){
        down();
        return key;
    }
};
node* merge(node* u, node* v){
    if(not u or not v) return u ?: v;
    u->down(); v->down();
    if(u->key.w > v->key.w) swap(u, v);
    swap(u->L, (u->R = merge(v, u->R)));
    return u;
}
void pop(node*& u){
    u->down();
    u = merge(u->L, u->R);
}
struct union_find : vector<int>{
    vector<pair<int, int>> st;
    union_find(int n) : vector<int>(n, -1){}
    int time(){return st.size();}
    int find(int u){return at(u) < 0 ? u : find(at(u));}
    void roll_back(int t){
        for(int i = time() - 1; i >= t; i -= 1)
            at(st[i].first) = st[i].second;
        st.resize(t);
    }
    int unite(int u, int v){
        u = find(u);
        v = find(v);
        if(u == v) return 0;
        if(at(u) > at(v)) std::swap(u, v);
        st.push_back({u, at(u)});
        st.push_back({v, at(v)});
        at(v) += at(u);
        at(u) = v;
        return 1;
    }
};
pair<LL, vector<int>> dmst(int n, const vector<Edge>& edges, int r){
    union_find uf(n);
    vector<node*> heap(n);
    for(Edge e : edges) heap[e.v] = merge(heap[e.v], new node(e));
    LL res = 0;
    vector<int> vis(n, -1), path(n), p(n);
    p[r] = vis[r] = r;
    vector<Edge> q(n), in(n, {-1, -1, 0});
    deque<tuple<int, int, vector<Edge>>> cycs;
    for(int i = 0; i < n; i += 1){
        int u = i, v = 0, qi = 0;
        while(vis[u] == -1){
            if(not heap[u]) return {-1, {}};
            Edge e = heap[u]->top();
            heap[u]->delta -= e.w;
            pop(heap[u]);
            q[qi] = e;
```

```
            path[qi ++] = u;
            vis[u] = i;
            res += e.w;
            u = uf.find(e.u);
            if(vis[u] == i){
                node* cyc = nullptr;
                int end = qi, time = uf.time();
                do cyc = merge(cyc, heap[v = path[qi -= 1]]);
                while(uf.unite(u, v));
                u = uf.find(u);
                heap[u] = cyc;
                vis[u] = -1;
                cycs.push_front({u, time, {&q[qi], &q[end]}});
            }
        }
        for(int i = 0; i < qi; i += 1) in[uf.find(q[i].v)] = q[i];
    }
    for(auto& [u, t, comp] : cycs){
        uf.roll_back(t);
        Edge in_edge = in[u];
        for(auto& e : comp) in[uf.find(e.v)] = e;
        in[uf.find(in_edge.v)] = in_edge;
    }
    for(int i = 0; i < n; i += 1) if(i != r) p[i] = in[i].u;
    return {res, p};
}
```

3.7 Flow and Matching

3.7.1 Maximum Flow(Dinic)

Time complexity: $O(n^2m)$.

```
struct Edge{
    int u, v;
    LL c;
};
LL Dinic(int n, const vector<Edge>& edges, int s, int t){
    vector<vector<int>> G(n);
    vector<vector<int>::const_iterator> cur(n);
    vector<Edge> e;
    vector<int> d(n);
    for(int i = 0; i < (int)edges.size(); i += 1){
        G[edges[i].u].push_back(e.size());
        e.push_back(edges[i]);
        G[edges[i].v].push_back(e.size());
        e.push_back({edges[i].v, edges[i].u, 0});
    }
    auto bfs = [&](){
        fill(d.begin(), d.end(), -1);
        queue<int> q;
        d[s] = 0;
        q.push(s);
        while(not q.empty()){
            int u = q.front();
            q.pop();
            for(int i : G[u])
                if(e[i].c and d[e[i].v] == -1){
                    d[e[i].v] = d[u] + 1;
                    q.push(e[i].v);
                }
        }
        return d[t] != -1;
    };
    function<LL(int, LL)> dfs = [&](int u, LL f){
        if(u == t) return f;
        LL ret = 0;
        for(; cur[u] != G[u].end(); cur[u] = next(cur[u])){
            int i = *cur[u];
            if(d[e[i].v] != d[u] + 1 or not e[i].c) continue;
            LL pf = dfs(e[i].v, min(e[i].c, f));
            e[i].c -= pf;
            e[i ^ 1].c += pf;
            ret += pf;
            f -= pf;
            if(not f) break;
        }
        return ret;
    };
    LL ret = 0;
    while(bfs()){
        for(int i = 0; i < n; i += 1) cur[i] = G[i].begin();
        ret += dfs(s, LLONG_MAX);
    }
    return ret;
}
```

3.7.2 Maximum Flow(Push Relabel)

Time complexity: $O(n^2\sqrt{m})$.

```
struct Edge{
    int u, v;
    LL c;
};
LL max_flow(int n, const vector<Edge>& edges, int s, int t){
    vector<vector<int>> G(n), H(n * 2);
    vector<Edge> e;
    vector<int> h(n), cur(n), ch(n * 2);
    vector<LL> p(n);
    for(auto [u, v, c] : edges) if(u != v){
        G[u].push_back(e.size());
        e.push_back({u, v, c});
        G[v].push_back(e.size());
        e.push_back({v, u, 0});
    }
    auto push = [&](int i, LL f){
        if(not p[e[i].v] and f) H[h[e[i].v]].push_back(e[i].v);
        e[i].c -= f;
        e[i ^ 1].c += f;
        p[e[i].u] -= f;
        p[e[i].v] += f;
    };
    h[s] = n;
    ch[0] = n - 1;
    p[t] = 1;
    for(int i : G[s]) push(i, e[i].c);
    for(int hi = 0;;){
        while(H[hi].empty()) if(not hi --) return -p[s];
        int u = H[hi].back();
        H[hi].pop_back();
        while(p[u] > 0){
            if(cur[u] == (int)G[u].size()){
                h[u] = INT_MAX;
                for(int& i : G[u]) if(e[i].c and h[u] > h[e[i].v] + 1){
                    h[u] = h[e[i].v] + 1;
                    cur[u] = &i - G[u].data();
                }
                ch[h[u]] += 1;
                if(not(ch[hi] -= 1) and hi < n)
                    for(int i = 0; i < n; i += 1)
                        if(h[i] > hi and h[i] < n){
                            ch[h[i]] -= 1;
                            h[i] = n + 1;
                        }
                hi = h[u];
            }
            else{
                int i = G[u][cur[u]];
                if(e[i].c and h[u] == h[e[i].v] + 1) push(i, min(p[u], e[i].c));
                else cur[u] += 1;
            }
        }
    }
    return 0LL;
}
```

3.7.3 Minimum Cost Maximum Flow(Dijkstra)

```
struct Edge{
    int u, v;
    LL cost, cap, flow;
};
pair<LL, LL> successive_shortest_path(int n, const vector<Edge>& E, int s, int t){
    vector<vector<int>> G(n);
    vector<Edge> edges;
    vector<LL> h(n), d(n);
    vector<int> p(n), done(n);
    vector<vector<int>::iterator> nxt(n);
    for(auto e : E){
        G[e.u].push_back(edges.size());
        edges.push_back(e);
        G[e.v].push_back(edges.size());
        edges.push_back({e.v, e.u, -e.cost, 0, 0});
    }
    auto dijkstra = [&](){
        fill(d.begin(), d.end(), 1E18);
        fill(p.begin(), p.end(), -1);
        fill(done.begin(), done.end(), 0);
        priority_queue<pair<LL, int>, vector<pair<LL, int>>, greater<pair<LL, int>>> q;
        q.push({d[s] = 0, s});
        while(not q.empty()){
            int u = q.top().second;
            q.pop();
            if(done[u]) continue;
```

```
            done[u] = 1;
            for(int i : G[u])
                if(edges[i].cap > edges[i].flow and d[edges[i].v] > d[u] + h[u] + edges[i].cost - h[edges[i].v]){
                    p[edges[i].v] = i;
                    q.push({d[edges[i].v] = d[u] + h[u] + edges[i].cost - h[edges[i].v], edges[i].v});
                }
        }
        return ~p[t];
    };
    LL f = 0, c = 0;
    while(dijkstra()){
        for(int i = 0; i < n; i += 1) h[i] += d[i];
        LL nf = LLONG_MAX;
        for(int u = t; u != s; u = edges[p[u]].u) nf = min(nf, edges[p[u]].cap - edges[p[u]].flow);
        f += nf;
        c += h[t] * nf;
        for(int u = t; u != s; u = edges[p[u]].u){
            edges[p[u]].flow += nf;
            edges[p[u] ^ 1].flow -= nf;
        }
    }
    return {f, c};
}
```

3.7.4 Bipartite Matching(Hopcroft Karp)

Time complexity: $O(m\sqrt{n})$.

```
vector<int> max_matching(int m, const vector<vector<int>>& G){
    int n = G.size();
    vector<int> A(n), B(m), res(m, -1), cur, next;
    while(true){
        fill(A.begin(), A.end(), 0);
        fill(B.begin(), B.end(), -1);
        cur.clear();
        for(int i : res) if(i != -1) A[i] = -1;
        for(int i = 0; i < n; i += 1) if(not A[i]) cur.push_back(i);
        for(int L = 1;; L += 2){
            bool isLast = false;
            next.clear();
            for(int i : cur) for(int j : G[i]){
                if(res[j] == -1){
                    B[j] = L;
                    isLast = true;
                }
                else if(res[j] != i and B[j] == -1){
                    B[j] = L;
                    next.push_back(res[j]);
                }
            }
            if(isLast) break;
            if(next.empty()) return res;
            for(int i : next) A[i] = L + 1;
            cur.swap(next);
        }
        function<bool(int, int)> dfs = [&](int u, int L){
            if(A[u] != L) return false;
            A[u] = -1;
            for(int v : G[u]) if(B[v] == L + 1){
                B[v] = -1;
                if(res[v] == -1 or dfs(res[v], L + 2))
                    return res[v] = u, true;
            }
            return false;
        };
        for(int i = 0; i < n; i += 1)
            dfs(i, 0);
    }
    return res;
}
```

3.7.5 Weighted Bipartite Matching(Kuhn Munkres)

Time complexity: $O(n^3)$.

```
vector<int> km(const vector<vector<LL>>& w) {
    int n = w.size();
    vector<LL> hl(n), hr(n);
    vector<int> fl(n, -1), fr(n, -1), pre(n);
    for (int i = 0; i < n; i += 1) hl[i] = *max_element(w[i].begin(), w[i].end());
    for (int s = 0; s < n; s += 1)
        [&](int s){
            vector<LL> slack(n, inf);
            vector<int> vl(n), vr(n);
            queue<int> q;
            q.push(s);
```

```
            vr[s] = 1;
            auto check = [&](int u) {
                vl[u] = 1;
                if (fl[u] != -1) {
                    q.push(fl[u]);
                    vr[fl[u]] = 1;
                    return 1;
                }
                while (u != -1) swap(u, fr[fl[u] = pre[u]]);
                return 0;
            };
            while (true) {
                while (not q.empty()) {
                    int u = q.front();
                    q.pop();
                    for (int i = 0; i < n; i += 1) {
                        LL d = hl[i] + hr[u] - w[i][u];
                        if (not vl[i] and slack[i] >= d) {
                            pre[i] = u;
                            if (d) slack[i] = d;
                            else if (not check(i)) return;
                        }
                    }
                }
                LL d = inf;
                for (int i = 0; i < n; i += 1)
                    if (not vl[i]) d = min(d, slack[i]);
                for (int i = 0; i < n; i += 1)
                    if (vl[i]) hl[i] += d;
                    else slack[i] -= d;
                for (int i = 0; i < n; i += 1) if (vr[i]) hr[i] -= d;
                for (int i = 0; i < n; i += 1)
                    if (not vl[i] and not slack[i] and not check(i))
                        return;
            }
        }(s);
    return fl;
}
```

3.7.6 General Matching(Blossom)

Time complexity: $O(n^3)$.

```
vector<int> matching(int n, const vector<pair<int, int>>& edges) {
    int time = 0;
    vector<int> matched(n, -1), pre(n, -1), vis(n);
    vector<vector<int>> G(n);
    for (auto [x, y] : edges) {
        G[x].push_back(y);
        G[y].push_back(x);
    }
    for (int i = 0; i < n; i += 1)
        if (matched[i] == -1)
            [&](int s) {
                vector<int> p(n), type(n, -1);
                for (int i = 0; i < n; i += 1) p[i] = i;
                function<int(int)> fp = [&](int u) {
                    return u == p[u] ? u : p[u] = fp(p[u]);
                };
                queue<int> q;
                auto push = [&](int u) {
                    q.push(u);
                    type[u] = 0;
                };
                push(s);
                while (not q.empty()) {
                    int u = q.front();
                    q.pop();
                    for (int v : G[u])
                        if (type[v] == -1) {
                            pre[v] = u, type[v] = 1;
                            if (matched[v] == -1) {
                                for (int x = u, y = v, tmp; x != -1; x = pre[y]) {
                                    tmp = matched[x], matched[x] = y, matched[y] = x;
                                    if ((y = tmp) == -1) break;
                                }
                                return;
                            }
                            push(matched[v]);
                        }
                        else if (not type[v] and fp(u) != fp(v)) {
                            int w = [&](int u, int v) {
                                for (time += 1, u = fp(u), v = fp(v); ; swap(u, v))
                                    if (~u) {
                                        if (vis[u] == time) return u;
                                        vis[u] = time;
                                        if (matched[u] == -1 or pre[matched[u]] == -1)
                                            u = -1;
                                        else u = fp(pre[matched[u]]);
```

```
                                }
                            }(u, v);
                            auto blossom = [&](int u, int v, int w) {
                                while (fp(u) != w) {
                                    pre[u] = v;
                                    v = matched[u];
                                    if (type[matched[u]] == 1) push(matched[u]);
                                    if (p[u] == u) p[u] = w;
                                    if (p[v] == v) p[v] = w;
                                    u = pre[v];
                                }
                            };
                            blossom(u, v, w);
                            blossom(v, u, w);
                        }
                    }
                }(i);
    return matched;
}
```

4 String

4.1 Knuth Morris Pratt

The length of longest border of $s_{0...i}$ is p_i.

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

4.2 Manacher

$s_{i-j} = s_{i+j}$ for $j < p_i$.

```
vector<int> manacher(const string& s){
  int n = s.size();
  vector<int> p(n);
  for(int i = 0, r = 0, m = 0; i < n; i += 1){
      p[i] = i < r ? min(p[m * 2 - i], r - i) : 1;
      while(i >= p[i] and i + p[i] < n and s[i - p[i]] == s[i + p[i]]) p[i] += 1;
      if(i + p[i] > r) m = i, r = i + p[i];
  }
  return p;
}
```

4.3 Z-Function

The length of longest common prefix of s and $s_{i...|s|-1}$ is z_i.

```
vector<int> z_function(string s) {
    int n = s.size();
    vector<int> z(n);
    for(int i = 1, l = 0, r = 0; i < n; i += 1){
        if(i <= r) z[i] = min(r - i + 1, z[i - l]);
        while(i + z[i] < n and s[z[i]] == s[i + z[i]]) z[i] += 1;
        if(i + z[i] - 1 > r){
            l = i;
            r = i + z[i] - 1;
        }
    }
    return z;
}
```

4.4 Lyndon Factorization

```
vector<int> duval(const vector<int>& s) {
    int n = s.size();
    vector<int> res;
    for(int i = 0, j, k; i < n; ) {
        j = i + 1, k = i;
        for(; j < n and s[k] <= s[j]; j += 1)
```

```
            if(s[k] < s[j]) k = i;
            else k += 1;
        for(; i <= k; i += j - k) res.push_back(i);
    }
    return res;
}
```

4.5 Suffix Array

```
struct Suffix_Array : vector<int> {
    vector<int> rank, lcp;
    Suffix_Array(const string& s) : vector<int>(s.size()), rank(s.size()), lcp(s.size()){
        int n = s.size(), k = 128;
        vector<int> cnt(max(n, k), 0), p(n);
        for (int i = 0; i < n; i += 1) cnt[rank[i] = s[i]] += 1;
        for (int i = 1; i < k; i += 1) cnt[i] += cnt[i - 1];
        for (int i = n - 1; i >= 0; i -= 1) at(cnt[rank[i]] -= 1) = i;
        for (int h = 1; h <= n; h <<= 1) {
            fill(cnt.begin(), cnt.end(), 0);
            for (int i = 0; i < n; i += 1) cnt[rank[i]] += 1;
            for (int i = 1; i < k; i += 1) cnt[i] += cnt[i - 1];
            k = 0;
            for (int i = n - h; i < n; i += 1) p[k ++] = i;
            for (int i = 0; i < n; i += 1) if (at(i) >= h) p[k ++] = at(i) - h;
            for (int i = n - 1; i >= 0; i -= 1) at(cnt[rank[p[i]]] -= 1) = p[i];
            p.swap(rank);
            rank[at(0)] = 0;
            k = 1;
            for (int i = 1; i < n; i += 1) {
                if (p[at(i)] != p[at(i - 1)] or at(i - 1) + h >= n or p[at(i) + h] != p[at(i - 1) + h]) k += 1;
                rank[at(i)] = k - 1;
            }
            if (k == n) break;
        }
        for (int i = 0, k = 0; i < n; i += 1) {
            if (k) k -= 1;
            if (rank[i]) while (s[i + k] == s[at(rank[i] - 1) + k]) k += 1;
            lcp[rank[i]] = k;
        }
    }
};
```

4.6 Aho Corasick

```
constexpr int maxc = 26;
struct State{
    int link, next[maxc];
    State() : link(0){
        fill(next, next + maxc, 0);
    }
};
struct AC : vector<State>{
    AC(){
        emplace_back();
    }
    int insert(const vector<int>& s){
        int p = 0;
        for(int c : s){
            if(not at(p).next[c]){
                at(p).next[c] = size();
                emplace_back();
            }
            p = at(p).next[c];
        }
        return p;
    }
    void init(){
        queue<int> q;
        q.push(0);
        while(not q.empty()){
            int u = q.front();
            q.pop();
            for(int i = 0; i < maxc; i += 1){
                int &s = at(u).next[i];
                if(not s) s = at(at(u).link).next[i];
                else{
                    at(s).link = u ? at(at(u).link).next[i] : 0;
                    q.push(s);
                }
            }
        }
    }
};
```

4.7 Palindromic Automaton

```
constexpr int maxc = 26;
struct State{
    int sum, len, link, next[maxc];
    State(int len) : len(len){
        sum = link = 0;
        fill(next, next + maxc, 0);
    }
};
struct PAM : vector<State>{
    int last;
    vector<int> s;
    PAM() : last(0){
        emplace_back(0);
        emplace_back(-1);
        at(0).link = 1;
    }
    int get_link(int u, int i){
        while(i < at(u).len + 1 or s[i - at(u).len - 1] != s[i]) u = at(u).link;
        return u;
    }
    void extend(int i){
        int cur = get_link(last, i);
        if(not at(cur).next[s[i]]){
            int now = size();
            emplace_back(at(cur).len + 2);
            back().link = at(get_link(at(cur).link, i)).next[s[i]];
            back().sum = at(back().link).sum + 1;
            at(cur).next[s[i]] = now;
        }
        last = at(cur).next[s[i]];
    }
};
```

4.8 Suffix Automaton

```
constexpr int maxc = 26;
struct State{
    int len, link, next[maxc];
    State(int len) : len(len), link(-1){
        fill(next, next + maxc, -1);
    }
};
struct SAM : vector<State>{
    int last;
    SAM() : last(0){
        emplace_back(0);
    };
    void extend(int c){
        int cur = size();
        emplace_back(at(last).len + 1);
        int p = last;
        for(; ~p and at(p).next[c] == -1; p = at(p).link) at(p).next[c] = cur;
        if(p == -1) back().link = 0;
        else{
            int q = at(p).next[c];
            if(at(p).len + 1 == at(q).len) back().link = q;
            else{
                int clone = size();
                push_back(at(q));
                back().len = at(p).len + 1;
                for(; ~p and at(p).next[c] == q; p = at(p).link) at(p).next[c] = clone;
                at(q).link = at(cur).link = clone;
            }
        }
        last = cur;
    }
};
```

5 Geometry

5.1 Convex Hull

```
struct P{
    R x, y;
    P operator - (const P& p){return {x - p.x, y - p.y};}
    R cross(const P& p){return x * p.y - y * p.x;}
};
vector<P> convex_hull(vector<P>& p){
    sort(p.begin(), p.end(), [](const P& A, const P& B){
        return A.x == B.x ? A.y < B.y : A.x < B.x;
```

```
    });
    vector<P> h;
    for(auto cur : p){
        while(h.size() >= 2 and (cur - h[h.size() - 2]).cross(h.back() - h[h.size() - 2]) >= 0) h.pop_back();
        h.push_back(cur);
    }
    int tmp = h.size();
    reverse(p.begin(), p.end());
    for(auto cur : p){
        while(h.size() - tmp >= 1 and (cur - h[h.size() - 2]).cross(h.back() - h[h.size() - 2]) >= 0) h.pop_back();
        h.push_back(cur);
    }
    h.pop_back();
    return h;
}
```

5.2 Halfplane Intersection(TBD)

```
// Redefine epsilon and infinity as necessary. Be mindful of precision errors.
const long double eps = 1e-9, inf = 1e9;

// Basic point/vector struct.
struct Point {

    long double x, y;
    explicit Point(long double x = 0, long double y = 0) : x(x), y(y) {}

    // Addition, substraction, multiply by constant, cross product.

    friend Point operator + (const Point& p, const Point& q) {
        return Point(p.x + q.x, p.y + q.y);
    }

    friend Point operator - (const Point& p, const Point& q) {
        return Point(p.x - q.x, p.y - q.y);
    }

    friend Point operator * (const Point& p, const long double& k) {
        return Point(p.x * k, p.y * k);
    }

    friend long double cross(const Point& p, const Point& q) {
        return p.x * q.y - p.y * q.x;
    }
};

// Basic half-plane struct.
struct Halfplane {

    // 'p' is a passing point of the line and 'pq' is the direction vector of the line.
    Point p, pq;
    long double angle;

    Halfplane() {}
    Halfplane(const Point& a, const Point& b) : p(a), pq(b - a) {
        angle = atan2l(pq.y, pq.x);
    }

    // Check if point 'r' is outside this half-plane.
    // Every half-plane allows the region to the LEFT of its line.
    bool out(const Point& r) {
        return cross(pq, r - p) < -eps;
    }

    // Comparator for sorting.
    // If the angle of both half-planes is equal, the leftmost one should go first.
    bool operator < (const Halfplane& e) const {
        if (fabsl(angle - e.angle) < eps) return cross(pq, e.p - p) < 0;
        return angle < e.angle;
    }

    // We use equal comparator for std::unique to easily remove parallel half-planes.
    bool operator == (const Halfplane& e) const {
        return fabsl(angle - e.angle) < eps;
    }

    // Intersection point of the lines of two half-planes. It is assumed they're never parallel.
    friend Point inter(const Halfplane& s, const Halfplane& t) {
        long double alpha = cross((t.p - s.p), t.pq) / cross(s.pq, t.pq);
        return s.p + (s.pq * alpha);
    }
};
ack(aux);
    }

    // Sort and remove duplicates
    sort(H.begin(), H.end());
    H.erase(unique(H.begin(), H.end()), H.end());
```

```
    deque<Halfplane> dq;
    int len = 0;
    for(int i = 0; i < int(H.size()); i++) {

        // Remove from the back of the deque while last half-plane is redundant
        while (len > 1 && H[i].out(inter(dq[len-1], dq[len-2]))) {
            dq.pop_back();
            --len;
        }

        // Remove from the front of the deque while first half-plane is redundant
        while (len > 1 && H[i].out(inter(dq[0], dq[1]))) {
            dq.pop_front();
            --len;
        }

        // Add new half-plane
        dq.push_back(H[i]);
        ++len;
    }

    // Final cleanup: Check half-planes at the front against the back and vice-versa
    while (len > 2 && dq[0].out(inter(dq[len-1], dq[len-2]))) {
        dq.pop_back();
        --len;
    }

    while (len > 2 && dq[len-1].out(inter(dq[0], dq[1]))) {
        dq.pop_front();
        --len;
    }

    // Report empty intersection if necessary
    if (len < 3) return vector<Point>();

    // Reconstruct the convex polygon from the remaining half-planes.
    vector<Point> ret(len);
    for(int i = 0; i+1 < len; i++) {
        ret[i] = inter(dq[i], dq[i+1]);
    }
    ret.back() = inter(dq[len-1], dq[0]);
    return ret;
}
```

6 Number Theory

6.1 Modular Arithmetic

6.1.1 Multiplication

```
LL mul(LL a, LL b, LL p){
    return (a * b - (LL)(a / (long double)p * b + 1e-3) * p + p) % p;
}
```

6.1.2 Sqrt(Cipolla)

p is prime.

```
struct P{
    LL x, y;
};
LL sqrt(LL n, LL p){
    if(n == 0) return 0;
    if(power(n, (p ^ 1) >> 1, p) != 1) return -1;
    LL x, w;
    do x = rand(), w = (x * x + p - n) % p;
    while(power(w, (p ^ 1) >> 1, p) == 1);
    auto mul = [&](P A, P B)->P{
        return {(A.x * B.x + A.y * B.y % p * w) % p, (A.x * B.y + A.y * B.x) % p};
    };
    P a = {x, 1}, res = {1, 0};
    for(LL r = (p + 1) >> 1; r; r >>= 1, a = mul(a, a))
        if(r & 1) res = mul(res, a);
    return res.x;
}
```

6.1.3 Log(Baby-Step Giant-Step)

```cpp
LL log(LL a, LL b, LL p){
    LL res = 0, k = 1, d;
    if(b == 1) return 0;
    if(not a) return b ? -1 : 1;
    for(;k != b and (d = gcd(a, p)) != 1; res += 1){
        if(b % d) return -1;
        p /= d;
        b /= d;
        k = k * (a / d) % p;
    }
    if(k == b) return res;
    unordered_map<LL, LL> mp;
    LL x = 1, y, M = sqrt(p) + 1;
    for(int i = 0; i < M; i += 1, x = x * a % p) mp[b * x % p] = i;
    y = k * x % p;
    for(int i = 1; i <= M; i += 1, y = y * x % p)
        if(mp.count(y)) return res + i * M - mp[y];
    return -1;
}
```

6.2 Miller Rabin

```cpp
bool miller_rabin(LL n){
    static LL p[9] = {2, 3, 5, 7, 11, 13, 17, 19, 23};
    if(n == 1) return false;
    if(n == 2) return true;
    if(not(n & 1)) return false;
    LL d = n - 1, r = 0;
    for(; not(d & 1); d >>= 1) r += 1;
    bool res = true;
    for(int i = 0; i < 9 and p[i] < n and res; i += 1){
        LL x = power(p[i], d, n);
        if(x == 1 or x == n - 1) continue;
        for(int j = 1; j < r; j += 1){
            x = mul(x, x, n);
            if(x == n - 1) break;
        }
        if(x != n - 1) res = false;
    }
    return res;
};
```

6.3 Pollard Rho

```cpp
void pollard_rho(LL n){
    if(n == 1) return;
    if(miller_rabin(n)) return;//find a prime factor
    LL d = n;
    while(d == n){
        d = 1;
        for(LL k = 1, y = 0, x = 0, s = 1, c = rand() % n; d == 1; k <<= 1, y = x, s = 1)
            for(int i = 1; i <= k; i += 1){
                x = (mul(x, x, n) + c) % n;
                s = mul(s, abs(x - y), n);
                if(not(i % 127) or i == k){
                    d = gcd(s, n);
                    if(d != 1) break;
                }
            }
    }
    pollard_rho(d);
    pollard_rho(n / d);
};
```

6.4 Extended Euclid

```cpp
LL exgcd(LL a, LL b, LL& x, LL& y){
    if(not b) return x = 1, y = 0, a;
    LL d = exgcd(b, a % b, x, y), t = x;
    return x = y, y = t - a / b * y, d;
};
```

6.5 Chinese Remainder Theorem

```cpp
pair<LL, LL> crt(const vector<pair<LL, LL>>& p){
    __int128 A = 1, B = 0;
    for(auto [a, b] : p){
```

```
        LL x, y, d = exgcd(A, a, x, y);
        if((b - B) % d) return {-1, -1};
        B += (b - B) / d * x % (a / d) * A;
        A = A / d * a;
        B = (B % A + A) % A;
    }
    return {A, B};
}
```

7 Numerical

7.1 Matrix Inverse and Rank

$A^{-1} = A^{-1}(2I - AA^{-1}) \pmod{p^k}$.

```
int matrix_inverse(vector<vector<LL>>& A){
    int n = A.size(), rank = 0;
    vector<vector<LL>> B(n, vector<LL>(n));
    for(int i = 0; i < n; i += 1) B[i][i] = 1;
    for(int i = 0, r = 0; i < n; i += 1){
        int k = -1;
        for(int j = r; j < n; j += 1) if(A[j][i]){
            k = j;
            break;
        }
        if(k == -1) continue;
        swap(A[r], A[k]);
        swap(B[r], B[k]);
        LL inv = power(A[r][i], mod - 2);
        for(int j = i; j < n; j += 1) A[r][j] = A[r][j] * inv % mod;
        for(int j = 0; j < n; j += 1) B[r][j] = B[r][j] * inv % mod;
        for(int j = 0; j < n; j += 1) if(j != r){
            LL c = A[j][i];
            for(int k = i; k < n; k += 1) A[j][k] = (A[j][k] + mod - A[r][k] * c % mod) % mod;
            for(int k = 0; k < n; k += 1) B[j][k] = (B[j][k] + mod - B[r][k] * c % mod) % mod;
        }
        rank += 1;
        r += 1;
    }
    for(int i = 0; i < n; i += 1)
        for(int j = 0; j < n; j += 1)
            A[i][j] = B[i][j];
    return rank;
}
```

7.2 Golden Section Search

```
constexpr LD Phi = (sqrt(5) - 1) / 2;
LD ternary_search(LD f(LD), LD L, LD R){
        LD mL = Phi * L + (1 - Phi) * R;
        LD mR = Phi * R + (1 - Phi) * L;
        LD fmL = f(mL), fmR = f(mR);
        for(int i = 0; i < step; i += 1)
                if(fmL > fmR){
                        L = mL;
                        mL = mR;
                        fmL = fmR;
                        fmR = f(mR = Phi * R + (1 - Phi) * L);
                }
                else{
                        R = mR;
                        mR = mL;
                        fmR = fmL;
                        fmL = f(mL = Phi * L + (1 - Phi) * R);
                }
    return (mL + mR) / 2;
}
```

7.3 Simpson

```
LD simpson(function<LD(LD)> f, LD L, LD R){
    return (f(L) + f((L + R) / 2) * 4 + f(R)) * (R - L) / 6;
}
LD simpson(function<LD(LD)> f, LD L, LD R, LD eps){
    function<LD(LD, LD, LD, LD)> rec = [&](LD L, LD R, LD S, LD e){
        LD M = (L + R) / 2;
        LD S1 = simpson(f, L, M), S2 = simpson(f, M, R);
        if(abs(S1 + S2 - S) <= 15 * e or R - L <= eps)
            return S1 + S2;
        return rec(L, M, S1, e / 2)  + rec(M, R, S2, e / 2);
```

```
    };
    return rec(L, R, simpson(f, L, R), eps);
}
```

8 Magic

8.1 Polynomial

```
constexpr LL mod = 998244353;
constexpr LL g = 3;
vector<int> r;
struct Poly : vector<LL>{
    Poly(){}
    Poly(int n) : vector<LL>(n){}
    Poly(const initializer_list<LL>& list) : vector<LL>(list){}
    void dft(int n, bool inverse = false){
        if((int)r.size() != n){
            r.resize(n);
            r[1] = n >> 1;
            for(int i = 2; i < n; i += 1) r[i] = r[i >> 1] >> 1 | (i & 1 ? n >> 1 : 0);
        }
        resize(n);
        for(int i = 0; i < n; i += 1) if(i < r[i]) std::swap(at(i), at(r[i]));
        for(int d = 0; (1 << d) < n; d += 1){
            int m = 1 << d, m2 = m << 1;
            LL _w = power(inverse ? power(g, mod - 2) : g, (mod - 1) / m2);
            for(int i = 0; i < n; i += m2)
                for(int w = 1, j = 0; j < m; j += 1, w = w * _w % mod){
                    LL& x = at(i + j + m), &y = at(i + j), t = w * x % mod;
                    x = y - t;
                    if(x < 0) x += mod;
                    y += t;
                    if(y >= mod) y -= mod;
                }
        }
        if(inverse) for(int i = 0, inv = power(n, mod - 2); i < n; i += 1) at(i) = at(i) * inv % mod;
    }
    Poly operator * (const Poly& p)const{
        auto a = *this, b = p;
        int k = 1, n = size() + p.size() - 1;
        while(k < n) k <<= 1;
        a.dft(k);
        b.dft(k);
        for(int i = 0; i < k; i += 1) a[i] = a[i] * b[i] % mod;
        a.dft(k, true);
        a.resize(n);
        return a;
    }
    Poly inverse()const{
        Poly a = {power(at(0), mod - 2)};
        for(int n = 1; n < (int)size(); n <<= 1){
            int k = n << 2;
            auto b = *this, c = a;
            for(int i = n << 1; i < (int)b.size(); i += 1) b[i] = 0;
            b.dft(k);
            c.dft(k);
            for(int i = 0; i < k; i += 1) b[i] = b[i] * c[i] % mod * c[i] % mod;
            b.dft(k, true);
            a.resize(n << 1);
            for(int i = 0; i < (n << 1); i += 1) a[i] = (2 * a[i] + mod - b[i]) % mod;
        }
        a.resize(size());
        return a;
    }
    pair<Poly, Poly> operator / (const Poly& p){
        int n = size() - p.size() + 1;
        auto a = *this, b = p;
        reverse(a.begin(), a.end());
        reverse(b.begin(), b.end());
        a.resize(n);
        b.resize(n);
        auto q = a * b.inverse();
        q.resize(n);
        reverse(q.begin(), q.end());
        auto r = p * q;
        r.resize(p.size() - 1);
        for(int i = 0; i + 1 < (int)p.size(); i += 1){
            r[i] = at(i) - r[i];
            if(r[i] < 0) r[i] += mod;
        }
        return {q, r};
    }
    Poly log()const{
        int n = size();
        Poly a(n - 1);
        for(int i = 0; i + 1 < n; i += 1) a[i] = at(i + 1) * (i + 1) % mod;
        a = a * inverse();
```

```
        a.resize(n);
        for(int i = n - 1; i >= 0; i -= 1) a[i] = i ? a[i - 1] * power(i, mod - 2) % mod : 0;
        return a;
    }
    Poly exp()const{
        Poly a = {1};
        for(int n = 1; n < (int)size(); n <<= 1){
            int k = n << 2;
            auto b = a.log();
            b.resize(k);
            for(int i = 0; i < k; i += 1)
                b[i] = ((i < (int)size() ? at(i) : 0) + not i + mod - b[i]) % mod;
            a = a * b;
            a.resize(k);
        }
        a.resize(size());
        return a;
    }
    Poly mulT(const Poly& p)const{
        auto a = *this;
        reverse(a.begin(), a.end());
        a = a * p;
        a.resize(size());
        reverse(a.begin(), a.end());
        return a;
    }
#define tm ((tl + tr) >> 1)
#define ls (v << 1)
#define rs (ls | 1)
    Poly eval(const Poly& x)const{
        vector<Poly> Q(x.size() << 2), P(x.size() << 2);
        Poly y(x.size());
        function<void(int, int, int)> dfs1 = [&](int v, int tl, int tr){
            if(tl == tr){
                Q[v].push_back(1);
                Q[v].push_back((mod - x[tm]) % mod);
                return;
            }
            dfs1(ls, tl, tm);
            dfs1(rs, tm + 1, tr);
            Q[v] = Q[ls] * Q[rs];
        };
        function<void(int, int, int)> dfs2 = [&](int v, int tl, int tr){
            if(tl == tr){
                y[tm] = P[v][0];
                return;
            }
            P[v].resize(tr - tl + 1);
            P[ls] = P[v].mulT(Q[rs]);
            P[rs] = P[v].mulT(Q[ls]);
            dfs2(ls, tl, tm);
            dfs2(rs, tm + 1, tr);
        };
        dfs1(1, 0, x.size() - 1);
        Q[1].resize(max(size(), x.size()));
        P[1] = mulT(Q[1].inverse());
        dfs2(1, 0, x.size() - 1);
        return y;
    }
    friend Poly inter(const Poly& x, const Poly& y) {
        vector<Poly> Q(x.size() << 2), P(x.size() << 2);
        function<void(int, int, int)> dfs1 = [&](int v, int tl, int tr){
            if(tl == tr){
                Q[v].push_back((mod - x[tm]) % mod);
                Q[v].push_back(1);
                return;
            }
            dfs1(ls, tl, tm);
            dfs1(rs, tm + 1, tr);
            Q[v] = Q[ls] * Q[rs];
        };
        dfs1(1, 0, x.size() - 1);
        Poly f((int)Q[1].size() - 1);
        for (int i = 0; i + 1 < Q[1].size(); i += 1) f[i] = (Q[1][i + 1] * (i + 1)) % mod;
        Poly g = f.eval(x);
        function<void(int, int, int)> dfs2 = [&](int v, int tl, int tr){
            if(tl == tr){
                P[v].push_back(y[tm] * power(g[tm], mod - 2) % mod);
                return;
            }
            dfs2(ls, tl, tm);
            dfs2(rs, tm + 1, tr);
            P[v].resize(tr - tl + 1);
            Poly A = P[ls] * Q[rs];
            Poly B = P[rs] * Q[ls];
            for (int i = 0; i <= tr - tl; i += 1) P[v][i] = (A[i] + B[i]) % mod;
        };
        dfs2(1, 0, x.size() - 1);
        return P[1];
    }
};
```

8.2 Fast Walsh Transform

```cpp
void fwt(vector<LL>& v, int inverse = false){
    int n = v.size();
    for(int i = 1; i < n; i <<= 1)
        for(int j = 0; j < n; j += i << 1)
            for(int k = 0; k < i; k += 1){
                LL& x = v[j + k], &y = v[i + j + k];
                tie(x, y) = inverse ? make_pair((x + y) / 2, (x - y) / 2) : make_pair(x + y, x - y);//xor
                x = inverse ? x - y : x + y;//and
                y = inverse ? y - x : x + y;//or
            }
}

void fwt(vector<LL>& v, int inverse = false){
    int n = v.size();
    for(int i = 1; i < n; i <<= 1)
        for(int j = 0; j < n; j += i << 1)
            for(int k = 0; k < i; k += 1){
                LL& x = v[j + k], &y = v[i + j + k];
                tie(x, y) = inverse ? make_pair((x + y) / 2, (x - y) / 2) : make_pair(x + y, x - y);//xor
                x = inverse ? x - y : x + y;//and
                y = inverse ? y - x : x + y;//or
            }
}

constexpr int k, w1;
void fwt(vector<LL>& v, int n, bool inverse = false){
    vector<LL> t(k), w(k);
    for(int i = 0; i < k; i += 1) w[i] = i ? w[i - 1] * w1 % mod : 1;
    for(int i = 1; i < n; i *= k){
        for(int l = 0; l < n; l += i * k){
            for(int j = l; j < l + i; j += 1){
                for(int a = 0; a < k; a += 1)
                    for(int b = t[a] = 0; b < k; b += 1)
                        t[a] = (t[a] + v[j + b * i] * w[b * (k + (inverse ? a : -a)) % k]) %  mod;
                for(int a = 0; a < k; a += 1) v[j + a * i] = t[a];
            }
        }
    }
        if(inverse){
        LL inv = power(n, mod - 2);
        for(int i = 0; i < n; i += 1) F[i] = F[i] * inv % mod;
    }
}
```