University of Latvia

“Skrupulozās zemenītes” (LU)

- Valters Kalniņš
- Kristaps Štāls
- Matīss Kristiņš

Contents

1. C++

1.1. Optimizations

```
#pragma GCC optimize("Ofast, unroll-loops")
#pragma GCC target("avx2,bmi,bmi2,lzcnt,popcnt,tune=native")
```

1.2. Hash function

```
static uint64_t splitmix64(uint64_t x)
{x+=0x9e3779b97f4a7c15;x=(x^(x>>30))*0xbf58476d1ce4e5b9;
x=(x^(x>>27))*0x94d049bb133111eb;
return x^(x>>31);}
struct custom_hash {size_t operator()(uint64_t x) const {
    static const uint64_t FIXED_RANDOM =
chrono::steady_clock::now().time_since_epoch().count();return
splitmix64(x+FIXED_RANDOM);}};
const long long mod=998244353;
//1000000007
long long modpow(long long n, long long m){long long
res=1;while(m){if(m&1)res=res*n%mod;n=n*n%mod;m>>=1;}return res;}
```

1.3. Bitset

```
bitset<10> bb("1010000000"); // reverse order constructor
cout << bb.count() << "\n"; // 2
cout << bb._Find_first() << "\n"; // 7
bb[0] = 1;
cout << bb._Find_first() << "\n"; // 0
```

1.4. C++ random

```
mt19937
rng(chrono::steady_clock::now().time_since_epoch().count());
```

2. Algebra

$$\sum_{i=1}^{n} k^2 = \frac{n(n+1)(2n+1)}{6}$$

$$\sum_{i=1}^{n} k^3 = \left(\frac{n(n+1)}{2}\right)^2$$

3. Number Theory

3.1. Rabin-Miller

```
using u64 = uint64_t;
using u128 = __uint128_t;

u64 binpower(u64 base, u64 e, u64 mod) {
    u64 result = 1;
    base %= mod;
    while (e) {
        if (e & 1)
            result = (u128)result * base % mod;
        base = (u128)base * base % mod;
        e >>= 1;
    }
    return result;
}

bool check_composite(u64 n, u64 a, u64 d, int s) {
    u64 x = binpower(a, d, n);
    if (x == 1 || x == n - 1)
        return false;
    for (int r = 1; r < s; r++) {
        x = (u128)x * x % n;
        if (x == n - 1)
            return false;
    }
    return true;
};

bool MillerRabin(u64 n, int iter=5) { // returns true if n is
probably prime, else returns false.
```

```
    if (n < 4)
        return n == 2 || n == 3;

    int s = 0;
    u64 d = n - 1;
    while ((d & 1) == 0) {
        d >>= 1;
        s++;
    }

    for (int i = 0; i < iter; i++) {
        int a = 2 + rand() % (n - 3);
        if (check_composite(n, a, d, s))
            return false;
    }
    return true;
}
```

3.2. Extended GCD

```
int gcd(int a, int b, int& x, int& y) {
    if (b == 0) {
        x = 1;
        y = 0;
        return a;
    }
    int x1, y1;
    int d = gcd(b, a % b, x1, y1);
    x = y1;
    y = x1 - y1 * (a / b);
    return d;
}
```

3.3. Chinese Remainder Theorem

Notes:

- Assumes all modulo are pairwise coprime
- If not, splitting modulos using prime powers works

```
int mod_inv(int a, int mod){
    int x, y;
    int g = extGcd(a, mod, x, y);
    x = (x % mod + mod) % mod;
    return x;
}

pair<int, int> crt(vector<pair<int, int>> congurences){
    // {mod, remainder}
    int M = 1;
    for(auto c : congurences){
        M *= c.first;
    }
    int solution = 0;
    for(auto c : congurences) {
        int a_i = c.second;
        int m_i = M / c.first;
        int n_i = mod_inv(m_i, c.first);
        solution = (solution + a_i * m_i % M * n_i) % M;
    }
    return {M, solution};
}
```

3.4. Random usable primes

666240077 964865333 115091077 378347773 568491163 295451837
658540403 856004729 843998543 380557313

3.5. Euler's totient function

Useful stuff: $a^{\varphi(n)} \equiv 1 (\bmod n)$ if $\gcd(a, n) = 1$

$\sum_{i|n} \varphi(i) = n$

```
phi[1] = 1;
for(ll i = 2; i <= n; i ++ ){
    phi[i] = i;
}
for(ll i = 2; i <= n ; i ++ ){
    if(pr[i] == false){
        for(ll j = i ; j <= n; j += i){
            phi[j] /= i;
            phi[j] *= (i - 1);
            pr[j] = true;
        }
    }
}
```

4. Combinatorics

4.1. Stars and bars

n balls, k boxes:

$$\binom{n+k-1}{k-1}$$

4.2. Vandermorde idenitity (and variants)

$\binom{m+n}{r} = \sum \binom{n}{k}\binom{n}{r-k}$

$\sum \binom{n}{x}\binom{m}{x} = \binom{n+m}{n}$

5. Data Structures

5.1. Treap

```
struct Node{
    int value, cnt, pri; Node *left, *right;
    Node(int p) : value(p), cnt(1), pri(gen()),
        left(NULL), right(NULL) {};
};
typedef Node* pnode;
int get(pnode q){if(!q) return 0; return q->cnt;}
void update_cnt(pnode &q){
    if(!q) return; q->cnt=get(q->left)+get(q->right)+1;
}
void merge(pnode &T, pnode lef, pnode rig){
    if(!lef){T=rig;return;} if(!rig){T=lef;return;}
    if(lef->pri>rig->pri){merge(lef->right,lef->right,rig);T=lef;
    }else{merge(rig->left, lef, rig->left); T = rig;}
    update_cnt(T);
}
void split(pnode cur, pnode &lef, pnode &rig, int key){
    if(!cur){lef=rig=NULL;return;} int id=get(cur->left)+1;
    if(id<=key){split(cur->right,cur->right,rig,key-id);lef=cur;}
    else {split(cur->left, lef, cur->left, key); rig = cur;}
    update_cnt(cur);
}
```

6. Graphs

6.1. k-shortest path

```
mt19937 mt(119);
template <typename T> using min_priority_queue =
priority_queue<T, vector<T>, greater<T>>;

template <typename T>
struct heap_node{
    array<heap_node*, 2> c;
    T key;
};

template <typename T>
heap_node<T>* insert(heap_node<T>* a, T new_key) {
    if(!a || new_key.first < a->key.first){
        heap_node<T>* n = new heap_node<T>;
        n->c = {a, nullptr};
        n->key = new_key;
        return n;
    }
    a = new heap_node<T>(*a);
    int z = mt() & 1;
    a->c[z] = insert(a->c[z], new_key);
    return a;
}

vector<ll> k_shortest_paths(int n, vector<pair<array<int, 2>,
ll>> edges, int st, int en, int K){
    int M = edges.size();
    vector<vector<tuple<int, int, ll>>> radj(n);
    for(int e = 0; e < M; e ++ ){
        auto [x, l] = edges[e];
        auto [u, v] = x;
        radj[v].push_back({e, u, l});
    }
    vector<ll> dist(n, -1);
    vector<int> prvE(n, -1);
    vector<int> toposort;
    {
        min_priority_queue<pair<ll, int>> pq;
        pq.push({dist[en] = 0, en});
        while(!pq.empty()){
            ll d = pq.top().first;
            int cur = pq.top().second;
            pq.pop();
            if(d > dist[cur]) continue;
            toposort.push_back(cur);
            // for(auto [e, nxt, l] : radj[cur]){
            for(auto ee : radj[cur]){
                int e = get<0>(ee);
                int nxt = get<1>(ee);
                int l = get<2>(ee);
                if(dist[nxt] == -1 || d + l < dist[nxt]){
                    prvE[nxt] = e;
```

```
                pq.push({dist[nxt] = d + l, nxt});
            }
        }
    }
}
vector<vector<pair<ll, int>>> adj(n);
for(int e = 0; e < M; e ++){
    auto& [x, l] = edges[e];
    const auto& [u, v] = x;
    if(dist[v] == -1) continue;

    l += dist[v] - dist[u];

    if(e == prvE[u]) continue;
    adj[u].push_back({l, v});
}
for(int i = 0 ; i < n; i ++ ){
    sort(adj[i].begin(), adj[i].end());
    adj[i].push_back({-1, -1});
}
using iter_t = decltype(adj[0].begin());
using hnode = heap_node<pair<ll, iter_t>>;
vector<hnode*> node_roots(n, nullptr);
for(int cur : toposort){
    if(cur != en){
        int prv = edges[prvE[cur]].first[1];
        node_roots[cur] = node_roots[prv];
    } else {
        node_roots[cur] = nullptr;
    }
    const auto& [l, nxt] = adj[cur][0];
    if(nxt != -1){
        node_roots[cur] = insert(node_roots[cur], {l,
adj[cur].begin()});
    }
}
vector<pair<ll, int>> dummy_adj({{0, st}, {-1, -1}});
vector<ll> res; res.reserve(K);
min_priority_queue<tuple<ll, hnode*, iter_t>> q;
q.push({dist[st], nullptr, dummy_adj.begin()});
while(int(res.size()) < K && !q.empty()) {
    auto [l, start_heap, val_iter] = q.top(); q.pop();
    res.push_back(l);
    ll elen = val_iter->first;
    if(next(val_iter)->second != -1){
        q.push({l - elen + next(val_iter)->first, nullptr,
next(val_iter)});
    }
    if(start_heap){
        for(int z = 0; z < 2 ; z ++ ){
            auto nxt_start = start_heap->c[z];
            if(!nxt_start) continue;
            q.push({l - elen + nxt_start->key.first,
nxt_start, nxt_start->key.second});
        }
    }
    {
        int nxt = val_iter->second;
        auto nxt_start = node_roots[nxt];
        if(nxt_start) {
            q.push({l + nxt_start->key.first, nxt_start,
nxt_start->key.second});
        }
    }
}
return res;
}
```

7. Algoritms

7.1. Kuhn's algorithm

```
// node matching indexed 1-n with 1-m
const int N = ansus;
vector<int> g[N];
int mt[N], ind[N];
bool used[N];
bool kuhn(int u)
{
    if(used[u])
        return 0;
    used[u]=1;
    for(auto v:g[u])
    {
        if(mt[v]==-1||kuhn(mt[v]))
        {
            mt[v]=u;
            ind[u]=v;
            return 1;
        }
    }
    return 0;
}
int main()
{
    for(int i = 0;i<m;i++)
        mt[i]=-1;
    for(int i = 0;i<n;i++)
        ind[i]=-1;
    for(int run = 1;run;)
    {
        run=0;
        for(int i = 0;i<n;i++)
            used[i]=0;
        for(int i = 0;i<n;i++)
            if(ind[i]==-1&&kuhn(i))
                run=1;
    }
    // ind[u] = -1, ja nav matchots, citadi ind[u] = indekss no
otras komponentes
}
```

7.2. Hopcroft-Karp (Max Matching)

```
// Hopcroft-Karp maximal matching in O(E*sqrt(V))

struct HopcroftKarp {
    int n, m;
    const int NIL = 0, INF = INT_MAX;
    vector<int> pair_u, pair_v, dist;
    vector<vector<int>> adj;

    HopcroftKarp() {};
    HopcroftKarp(int n, int m) : n(n), m(m) {
        adj.resize(n+1);
    }

    // 1-indexed
    void add_edge(int u, int v) {adj[u].push_back(v);}

    int calc() {
        pair_u.assign(n+1, NIL);
        pair_v.assign(m+1, NIL);
        dist.assign(n+1, 0);
        int ans = 0;
        while (bfs()) {
            for (int u = 1; u <= n; u++) {
                if (pair_u[u] == NIL && dfs(u)) ans++;
            }
        }
        return ans;
    }

    bool bfs() {
        queue<int> q;
        for (int u = 1; u <= n; u++) {
            dist[u] = INF;
            if (pair_u[u] != NIL) continue;
            q.push(u);
            dist[u] = 0;
        }
        dist[NIL] = INF;
        while (!q.empty()) {
            int u = q.front();
            q.pop();
            if (dist[u] >= dist[NIL]) continue;
            for (auto &v : adj[u]) {
                if (dist[pair_v[v]] != INF) continue;
                dist[pair_v[v]] = dist[u] + 1;
                q.push(pair_v[v]);
            }
        }
        return (dist[NIL] != INF);
    }

    bool dfs(int u) {
        if (u == NIL) return true;
        for (auto &v : adj[u]) {
            if (dist[pair_v[v]] != dist[u] + 1 || !
dfs(pair_v[v])) continue;
            pair_v[v] = u;
            pair_u[u] = v;
            return true;
        }
        dist[u] = INF;
        return false;
    }
};
```

7.3. Hungarian Algorithm (Min cost, Max matching)

```
// Hungarian algorithm O(n^3) (but fast)
// 1-indexed
// It finds minimum cost maximum matching.
// For finding maximum cost maximum matching add -cost and return
-matching()
// matching stored in l array. l[i] contains index of right side
element that is match with the i-th left side element.
const int N = 1024;
struct Hungarian {
    ll c[N][N], fx[N], fy[N], d[N];
    int l[N], r[N], arg[N], trace[N];
    queue<int> q;
    int start, finish, n;
    const ll inf = 1e18;
    Hungarian() {}
    Hungarian(int n1, int n2) {init(n1, n2);}
    void init(int n1, int n2) {
        n = max(n1, n2);
        for (int i = 1; i <= n; ++i) {
            fy[i] = l[i] = r[i] = 0;
            for (int j = 1; j <= n; ++j) c[i][j] = inf;
        }
    }
    void add_edge(int u, int v, ll cost) {
        c[u][v] = min(c[u][v], cost);
    }
    inline ll getC(int u, int v) {
        return c[u][v] - fx[u] - fy[v];
    }
    void init_bfs() {
        while (!q.empty()) q.pop();
        q.push(start);
        for (int i = 0; i <= n; ++i) trace[i] = 0;
        for (int v = 1; v <= n; ++v) {
            d[v] = getC(start, v);
            arg[v] = start;
        }
        finish = 0;
    }
    void find_aug_path() {
        while (!q.empty()) {
            int u = q.front(); q.pop();
            for (int v = 1; v <= n; ++v) if (!trace[v]) {
                ll w = getC(u, v);
                if (!w) {
                    trace[v] = u;
                    if (!r[v]) {
                        finish = v;
                        return;
                    }
                    q.push(r[v]);
                }
                if (d[v] > w) {
                    d[v] = w;
                    arg[v] = u;
                }
            }
        }
    }
    void subX_addY() {
        ll delta = inf;
        for (int v = 1; v <= n; ++v) if (trace[v] == 0 && d[v] <
delta) {
            delta = d[v];
        }
        fx[start] += delta;
        for (int v = 1; v <= n; ++v) if(trace[v]) {
            int u = r[v];
            fy[v] -= delta;
            fx[u] += delta;
        } else d[v] -= delta;
        for (int v = 1; v <= n; ++v) if (!trace[v] && !d[v]) {
            trace[v] = arg[v];
            if (!r[v]) {
                finish = v; return;
            }
            q.push(r[v]);
        }
    }
    void enlarge() {
        do {
            int u = trace[finish];
            int nxt = l[u];
            l[u] = finish;
            r[finish] = u;
            finish = nxt;
        } while (finish);
    }
    ll maximum_matching() {
        for (int u = 1; u <= n; ++u) {
            fx[u] = c[u][1];
            for (int v = 1; v <= n; ++v) {
                fx[u] = min(fx[u], c[u][v]);
            }
        }
        for (int v = 1; v <= n; ++v) {
            fy[v] = c[1][v] - fx[1];
            for (int u = 1; u <= n; ++u) {
                fy[v] = min(fy[v], c[u][v] - fx[u]);
            }
        }
        for (int u = 1; u <= n; ++u) {
            start = u;
            init_bfs();
            while (!finish) {
                find_aug_path();
                if (!finish) subX_addY();
            }
            enlarge();
        }
        ll ans = 0;
        for (int i = 1; i <= n; ++i) {
            if (c[i][l[i]] != inf) ans += c[i][l[i]];
            else l[i] = 0;
        }
        return ans;
    }
};
```

8. Flows

8.1. Dinitz

```
struct FlowEdge {
    int v, u;
    ll cap, flow = 0;
    FlowEdge(int v, int u, ll cap) : v(v), u(u), cap(cap) {}
};

struct Dinic {
    const long long flow_inf = 1e18;
    vector<FlowEdge> edges;
    vector<vector<int>> adj;
    int n, m = 0;
    int s, t;
    vector<int> level, ptr;
    queue<int> q;
    Dinic(int n, int s, int t) : n(n), s(s), t(t) {
        adj.resize(n);
        level.resize(n);
        ptr.resize(n);
    }
    void add_edge(int v, int u, ll cap) {
        edges.push_back(v, u, cap);
        edges.push_back(u, v, 0);
        adj[v].push_back(m);
        adj[u].push_back(m + 1);
        m += 2;
    }
    bool bfs() {
        while (!q.empty()) {
            int v = q.front();
            q.pop();
            for (int id : adj[v]) {
                if (edges[id].cap - edges[id].flow < 1)
                    continue;
                if (level[edges[id].u] != -1)
                    continue;
                level[edges[id].u] = level[v] + 1;
                q.push(edges[id].u);
            }
        }
        return level[t] != -1;
    }
    ll dfs(int v, ll pushed) {
        if (pushed == 0)
            return 0;
        if (v == t)
            return pushed;
        for (int& cid = ptr[v]; cid < (int)adj[v].size(); cid++)
{
            int id = adj[v][cid];
            int u = edges[id].u;
            if (level[v] + 1 != level[u] || edges[id].cap -
edges[id].flow < 1)
                continue;
```

```
            ll tr = dfs(u, min(pushed, edges[id].cap -
edges[id].flow));
            if (tr == 0)
                continue;
            edges[id].flow += tr;
            edges[id ^ 1].flow -= tr;
            return tr;
        }
        return 0;
    }
    ll flow() {
        ll f = 0;
        while (true) {
            fill(level.begin(), level.end(), -1);
            level[s] = 0;
            q.push(s);
            if (!bfs())
                break;
            fill(ptr.begin(), ptr.end(), 0);
            while (ll pushed = dfs(s, flow_inf)) {
                f += pushed;
            }
        }
        return f;
    }
};
```

8.2. Minimum-cost Max-Flow

```
struct Edge
{
    int from;
    int to;
    int capacity;
    int cost;
};

vector<vector<int>> adj, cost, capacity;
const int inf = (int)1e9;

void shortest_paths(int n, int v0, vector<Edge> &edges,
vector<int>& d, vector<int>& p){
    d.assign(n, inf);
    d[v0] = 0;
    vector<bool> inq(n, false);
    queue<int> q;
    q.push(v0);
    p.assign(n, -1);
    while(!q.empty()){
        int u = q.front();
        q.pop();
        inq[u] = false;
        for(int v : adj[u]){
            if(edges[v].capacity > 0 && d[edges[v].to] > d[u] +
edges[v].cost){
                d[edges[v].to] = d[u] + edges[v].cost;
                p[edges[v].to] = v;
                if(!inq[edges[v].to]) {
                    inq[edges[v].to] = true;
                    q.push(edges[v].to);
                }
            }
        }
    }
}

int min_cost_flow(int n, vector<Edge> edges, int K, int s, int t)
{
    int m = 0;
    adj.resize(n);
    vector<Edge> edg;
    for(Edge e : edges){
        edg.push_back({e.from, e.to, e.capacity, e.cost});
        edg.push_back({e.to, e.from, 0, -e.cost});
        adj[e.from].push_back(m);
        adj[e.to].push_back(m + 1);
        m += 2;
    }
    int flow = 0;
    int cost = 0;
    vector<int> d, p;
    while(flow < K){
        shortest_paths(n, s, edg, d, p);
        if(d[t] == inf)
            break;
        int f = K - flow;
        int cur = t;
        while(cur != s){
            f = min(f, edg[p[cur]].capacity);
            cur = edg[p[cur]].from;
        }

        flow += f;
        cost += f * d[t];
        cur = t;
        while(cur != s){
            edg[p[cur]].capacity -= f;
            edg[p[cur]^1].capacity += f;
            cur = edg[p[cur]].from;
        }
    }
    if(flow < K)
        return -inf;
    else
        return cost;
}
```

9. Strings

9.1. Manacher's algorithm longest palindromic substring

```
int manacher(string s){
    int n = s.size(); string p = "^#";
    rep(i,0,n) p += string(1, s[i]) + "#";
    p += "$"; n = p.size(); vector<int> lps(n, 0);
    int C=0, R=0, m=0;
    rep(i,1,n-1){
        int mirr = 2*C - i;
        if(i < R) lps[i] = min(R-i, lps[mirr]);
        while(p[i + 1 + lps[i]] == p[i - 1 - lps[i]]) lps[i]++;
        if(i + lps[i] > R){ C = i; R = i + lps[i]; }
        m = max(m, lps[i]);
    }
    return m;
}
```

9.2. Palindromic Tree (eertree)

```
struct eertree{

    int nex[N][AL];
    int ret[N];
    int par[N];
    int len[N];

    int id;
    void init(){
        len[0] = -1;
        ret[0] = 0;

        len[1] = 0;
        ret[1] = 0;

        id = 2;
    }
    string s;
    int n;
    void construct(string _s){
        s = _s;
        n = s.size();
        int las = 1;
        for(int i = 0 ; i < n; i ++ ){
            int cur = las;
            int l = s[i] - 'a' + 1;
            while(i - len[cur] - 1 < 0 || s[i] != s[i - len[cur]
- 1]){
                cur = ret[cur];
            }
            if(nex[cur][l] == 0){
                nex[cur][l] = id;
                len[id] = len[cur] + 2;
                par[id] = cur;
                if(cur == 0){
                    ret[id] = 1;
                }
                else{
                    int w = ret[cur];
                    while(i - len[w] - 1 < 0 || s[i] != s[i -
len[w] - 1]){
                        w = ret[w];
                    }
                    ret[id] = nex[w][l];
                }
                id ++ ;
            }
            las = nex[cur][l];
        }
    }
```

```
};
```

9.3. Suffix Array

```
const int M = 26;

void count_sort(vector<int> &p, vector<int> &c)
{
    int n = p.size();
    vector<int> pos(M+1);
    for(auto x:c)
        pos[x+1]++;
    for(int i = 1;i<=M;i++)
        pos[i]+=pos[i-1];
    vector<int> p_new(n);
    for(int i = 0;i<n;i++)
        p_new[pos[c[p[i]]]++]=p[i];
    swap(p,p_new);
}
int main()
{
    fio
    //ifstream cin("in.in");
    int n, m;
    cin >> n >> m;
    vector<int> str(n);
    for(auto &x:str)
        cin >> x;
    str.pb(-1);
    n++;
    vector<int> p(n), c(n);
    {
        vector<pair<char,int> > ve(n);
        for(int i = 0;i<n;i++)
            ve[i]={str[i],i};
        sort(ve.begin(),ve.end());
        for(int i = 0;i<n;i++)
            p[i]=ve[i].se;
        for(int i = 1;i<n;i++)
            c[p[i]]=c[p[i-1]]+(ve[i].fi!=ve[i-1].fi);
    }
    for(int k = 0;(1<<k)<n;k++)
    {
        for(int i = 0;i<n;i++)
            p[i]=(p[i]-(1<<k)+n)%n;
        count_sort(p,c);
        vector<int> c_new(n);
        for(int i = 1;i<n;i++)
            c_new[p[i]]=c_new[p[i-1]]+(c[p[i]]!=c[p[i-1]]||
c[(p[i]+(1<<k))%n]!=c[(p[i-1]+(1<<k))%n]);
        swap(c,c_new);
    }
    vector<int> lcp(n);
    int k = 0;
    for(int i = 0;i<n-1;i++)
    {
        int j = p[c[i]-1];
        while(str[i+k]==str[j+k])
            k++;
        lcp[c[i]]=k;
        k=max(k-1,0);
    }
    return 0;
}
```

9.4. Suffix Array and LCP (MK)

```
vector<int> suffix_array(string s){
    int n = s.size();
    int alphabet = 256;
    vector<int> p(n), c(n), cnt(max(alphabet, n), 0);
    for(int i = 0 ; i < n; i ++ ){
        cnt[s[i]] ++ ;
    }
    for(int i = 1; i < cnt.size(); i ++ ){
        cnt[i] += cnt[i - 1];
    }
    for(int i = 0 ; i < n; i ++ ){
        cnt[s[i]] -- ;
        p[cnt[s[i]]]=i;
    } // order
    c[p[0]] = 0;
    int classes = 1;
    for(int i = 1; i < n; i ++ ){
        c[p[i]] = c[p[i - 1]];
        if(s[p[i]] != s[p[i - 1]]){
            classes ++  ;
        }
        c[p[i]] = classes - 1;
    }
    vector<int> pn(n), cn(n);
    for (int h = 0; (1 << h) < n; ++h) {
        for (int i = 0; i < n; i++) {
            pn[i] = p[i] - (1 << h);
            if (pn[i] < 0)
                pn[i] += n;
        }
        fill(cnt.begin(), cnt.begin() + classes, 0);
        for (int i = 0; i < n; i++)
            cnt[c[pn[i]]]++;
        for (int i = 1; i < classes; i++)
            cnt[i] += cnt[i-1];
        for (int i = n-1; i >= 0; i--)
            p[--cnt[c[pn[i]]]] = pn[i];
        cn[p[0]] = 0;
        classes = 1;
        for (int i = 1; i < n; i++) {
            pair<int, int> cur = {c[p[i]], c[(p[i] + (1 << h)) %
n]};
            pair<int, int> prev = {c[p[i-1]], c[(p[i-1] + (1 <<
h)) % n]};
            if (cur != prev)
                ++classes;
            cn[p[i]] = classes - 1;
        }
        c.swap(cn);
    }
    return p;
}

vector<int> lcp_construct(string s, vector<int> p){
    int n = s.size();
    vector<int> rank(n, 0);
    for (int i = 0; i < n; i++)
        rank[p[i]] = i;

    int k = 0;
    vector<int> lcp(n-1, 0);
    for (int i = 0; i < n; i++) {
        if (rank[i] == n - 1) {
            k = 0;
            continue;
        }
        int j = p[rank[i] + 1];
        while (i + k < n && j + k < n && s[i+k] == s[j+k])
            k++;
        lcp[rank[i]] = k;
        if (k)
            k--;
    }
    return lcp;
}

void baseline(string s){
    vector<int> suffix = suffix_array(s);
    suffix.erase(suffix.begin());
    s.pop_back();
    vector<int> lcp = lcp_construct(s, suffix);
}
```

9.5. Aho-Corasick

```
const int K = 26;

struct Vertex {
    int next[K];
    bool output = false;
    int p = -1;
    char pch;
    int link = -1;
    int go[K];

    Vertex(int p=-1, char ch='$') : p(p), pch(ch) {
        fill(begin(next), end(next), -1);
        fill(begin(go), end(go), -1);
    }
};

vector<Vertex> t(1);

void add_string(string const& s) {
    int v = 0;
    for (char ch : s) {
        int c = ch - 'a';
        if (t[v].next[c] == -1) {
            t[v].next[c] = t.size();
            t.emplace_back(v, ch);
        }
        v = t[v].next[c];
```

```
    }
    t[v].output = true;
}

int go(int v, char ch);

int get_link(int v) {
    if (t[v].link == -1) {
        if (v == 0 || t[v].p == 0)
            t[v].link = 0;
        else
            t[v].link = go(get_link(t[v].p), t[v].pch);
    }
    return t[v].link;
}

int go(int v, char ch) {
    int c = ch - 'a';
    if (t[v].go[c] == -1) {
        if (t[v].next[c] != -1)
            t[v].go[c] = t[v].next[c];
        else
            t[v].go[c] = v == 0 ? 0 : go(get_link(v), ch);
    }
    return t[v].go[c];
}
```

9.6. KMP

```
vector<int> prefix_function(string s) {
    int n = (int)s.length();
    vector<int> pi(n);
    for (int i = 1; i < n; i++) {
        int j = pi[i-1];
        while (j > 0 && s[i] != s[j])
            j = pi[j-1];
        if (s[i] == s[j])
            j++;
        pi[i] = j;
    }
    return pi;
}
```

9.7. Z-Function

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

10. Geometry

10.1. Point to Line

Line $(Ax + By + C = 0)$ and point $(x_0; y_0)$ distance is:

$$d = \frac{|Ax_0 + By_0 + C|}{\sqrt{A^2 + B^2}}$$

10.2. Graham scan

```
struct pt {
    double x, y;
    bool operator == (pt const& t) const {
        return x == t.x && y == t.y;
    }
};

int orientation(pt a, pt b, pt c) {
    double v = a.x*(b.y-c.y)+b.x*(c.y-a.y)+c.x*(a.y-b.y);
    if (v < 0) return -1; // clockwise
    if (v > 0) return +1; // counter-clockwise
    return 0;
}

bool cw(pt a, pt b, pt c, bool include_collinear) {
    int o = orientation(a, b, c);
    return o < 0 || (include_collinear && o == 0);
}
bool collinear(pt a, pt b, pt c) { return orientation(a, b, c) ==
0; }

void convex_hull(vector<pt>& a, bool include_collinear = false) {
    pt p0 = *min_element(a.begin(), a.end(), [](pt a, pt b) {
        return make_pair(a.y, a.x) < make_pair(b.y, b.x);
    });
    sort(a.begin(), a.end(), [&p0](const pt& a, const pt& b) {
        int o = orientation(p0, a, b);
        if (o == 0)
            return (p0.x-a.x)*(p0.x-a.x) + (p0.y-a.y)*(p0.y-a.y)
                < (p0.x-b.x)*(p0.x-b.x) + (p0.y-b.y)*(p0.y-b.y);
        return o < 0;
    });
    if (include_collinear) {
        int i = (int)a.size()-1;
        while (i >= 0 && collinear(p0, a[i], a.back())) i--;
        reverse(a.begin()+i+1, a.end());
    }

    vector<pt> st;
    for (int i = 0; i < (int)a.size(); i++) {
        while (st.size() > 1 && !cw(st[st.size()-2], st.back(),
a[i], include_collinear))
            st.pop_back();
        st.push_back(a[i]);
    }

    if (include_collinear == false && st.size() == 2 && st[0] ==
st[1])
        st.pop_back();

    a = st;
}
```

10.3. Cross Product in 2D space

$\vec{a} \circ \vec{b} = a_x b_y - a_y b_x$

10.4. Shoelace formula

$A = \frac{1}{2} \sum_{i=1}^{n} x_i (y_{i+1} - y_{i-1})$ (counter clock wise direction)

10.5. Online Convex Hull trick

```
// KTH notebook
struct Line {
  mutable ll k, m, p;
  bool operator<(const Line& o) const { return k < o.k; }
  bool operator<(ll x) const { return p < x; }
};

struct LineContainer : multiset<Line, less<>> {
  // (for doubles, use inf = 1/.0, div(a,b) = a/b)
  static const ll inf = LLONG_MAX;
  ll div(ll a, ll b) { // floored division
    return a / b - ((a ^ b) < 0 && a % b); }
  bool isect(iterator x, iterator y) {
    if (y == end()) return x->p = inf, 0;
    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) {
    auto z = insert({k, m, 0}), y = z++, x = y;
    while (isect(y, z)) z = erase(z);
    if (x != begin() && isect(--x, y)) isect(x, y = erase(y));
    while ((y = x) != begin() && (--x)->p >= y->p)
      isect(x, erase(y));
  }
  ll query(ll x) {
    assert(!empty());
    auto l = *lower_bound(x);
    return l.k * x + l.m;
  }
};
```

10.6. Maximum points in a circle of radius R

```
typedef pair<double,bool> pdb;

#define START 0
#define END 1

struct PT
{
  double x, y;
  PT() {}
  PT(double x, double y) : x(x), y(y) {}
  PT(const PT &p) : x(p.x), y(p.y)    {}
  PT operator + (const PT &p)  const { return PT(x+p.x, y+p.y); }
  PT operator - (const PT &p)  const { return PT(x-p.x, y-p.y); }
```

```
  PT operator * (double c)     const { return PT(x*c,   y*c  ); }
  PT operator / (double c)     const { return PT(x/c,   y/c  ); }
};

PT p[505];
double dist[505][505];
int n, m;

void calcDist()
{
  FOR(i,0,n)
  {
    FOR(j,i+1,n)
      dist[i][j]=dist[j][i]=sqrt((p[i].x-p[j].x)*(p[i].x-p[j].x)
        +(p[i].y-p[j].y)*(p[i].y-p[j].y));
  }
}
int intelInside(int point, double radius)
{
  vector<pdb> ranges;
  FOR(j,0,n)
  {
    if(j==point || dist[j][point]>2*radius) continue;
    double a1=atan2(p[point].y-p[j].y,p[point].x-p[j].x);
    double a2=acos(dist[point][j]/(2*radius));
    ranges.pb({a1-a2,START});
    ranges.pb({a1+a2,END});
  }
  sort(ALL(ranges));
  int cnt=1, ret=cnt;
  for(auto it: ranges)
  {
    if(it.second) cnt--;
    else cnt++;
    ret=max(ret,cnt);
  }

  return ret;
}

int go(double r)
{
  int cnt=0;
  FOR(i,0,n)
  {
    cnt=max(cnt,intelInside(i,r));
  }
  return cnt;
}
```

10.7. Point in polygon

```
int sideOf(const PT &s, const PT &e, const PT &p)
{
  ll a = cross(e-s,p-s);
  return (a > 0) - (a < 0);
}

bool onSegment(const PT &s, const PT &e, const PT &p)
{
  PT ds = p-s, de = p-e;
  return cross(ds,de) == 0 && dot(ds,de) <= 0;
}

/*
Main routine
Description: Determine whether a point t lies inside a given
polygon (counter-clockwise order).
The polygon must be such that every point on the circumference is
visible from the first point in the vector.
It returns 0 for points outside, 1 for points on the
circumference, and 2 for points inside.
*/

int insideHull2(const vector<PT> &H, int L, int R, const PT &p) {
  int len = R - L;
  if (len == 2) {
    int sa = sideOf(H[0], H[L], p);
    int sb = sideOf(H[L], H[L+1], p);
    int sc = sideOf(H[L+1], H[0], p);
    if (sa < 0 || sb < 0 || sc < 0) return 0;
    if (sb==0 || (sa==0 && L == 1) || (sc == 0 && R ==
(int)H.size()))
      return 1;
    return 2;
  }
  int mid = L + len / 2;
  if (sideOf(H[0], H[mid], p) >= 0)
    return insideHull2(H, mid, R, p);
  return insideHull2(H, L, mid+1, p);
}

int insideHull(const vector<PT> &hull, const PT &p) {
  if ((int)hull.size() < 3) return onSegment(hull[0],
hull.back(), p);
  else return insideHull2(hull, 1, (int)hull.size(), p);
}
```

10.8. Minkowski Sum

```
struct pt{
    long long x, y;
    pt operator + (const pt & p) const {
        return pt{x + p.x, y + p.y};
    }
    pt operator - (const pt & p) const {
        return pt{x - p.x, y - p.y};
    }
    long long cross(const pt & p) const {
        return x * p.y - y * p.x;
    }
};

void reorder_polygon(vector<pt> & P){
    size_t pos = 0;
    for(size_t i = 1; i < P.size(); i++){
        if(P[i].y < P[pos].y || (P[i].y == P[pos].y && P[i].x <
P[pos].x))
            pos = i;
    }
    rotate(P.begin(), P.begin() + pos, P.end());
}

vector<pt> minkowski(vector<pt> P, vector<pt> Q){
    // the first vertex must be the lowest
    reorder_polygon(P);
    reorder_polygon(Q);
    // we must ensure cyclic indexing
    P.push_back(P[0]);
    P.push_back(P[1]);
    Q.push_back(Q[0]);
    Q.push_back(Q[1]);
    // main part
    vector<pt> result;
    size_t i = 0, j = 0;
    while(i < P.size() - 2 || j < Q.size() - 2){
        result.push_back(P[i] + Q[j]);
        auto cross = (P[i + 1] - P[i]).cross(Q[j + 1] - Q[j]);
        if(cross >= 0 && i < P.size() - 2)
            ++i;
        if(cross <= 0 && j < Q.size() - 2)
            ++j;
    }
    return result;
}
```

11. Numerical

11.1. FFT

```
using cd = complex<double>;
const double PI = acos(-1);

void fft(vector<cd> & a, bool invert) {
    int n = a.size();
    for (int i = 1, j = 0; i < n; i++) {
        int bit = n >> 1;
        for (; j & bit; bit >>= 1)
            j ^= bit;
        j ^= bit;
        if (i < j)
            swap(a[i], a[j]);
    }
    for (int len = 2; len <= n; len <<= 1) {
        double ang = 2 * PI / len * (invert ? -1 : 1);
        cd wlen(cos(ang), sin(ang));
        for (int i = 0; i < n; i += len) {
            cd w(1);
            for (int j = 0; j < len / 2; j++) {
                cd u = a[i+j], v = a[i+j+len/2] * w;
                a[i+j] = u + v;
                a[i+j+len/2] = u - v;
                w *= wlen;
            }
        }
    }
    if (invert) {
        for (cd & x : a)
            x /= n;
    }
```

```
}
vector<int> multiply(vector<int> const& a, vector<int> const& b)
{
    vector<cd> fa(a.begin(), a.end()), fb(b.begin(), b.end());
    int n = 1;
    while (n < a.size() + b.size())
        n <<= 1;
    fa.resize(n);
    fb.resize(n);
    fft(fa, false);
    fft(fb, false);
    for (int i = 0; i < n; i++)
        fa[i] *= fb[i];
    fft(fa, true);
    vector<int> result(n);
    for (int i = 0; i < n; i++)
        result[i] = round(fa[i].real());
    return result;
}
```

11.2. NTT

```
const ll mod = (119 << 23) + 1, root = 62; // 998244353
typedef vector<ll> vl;

int modpow(int n, int k);

void ntt(vl &a) {
  int n = a.size(), L = 31 - __builtin_clz(n);
  static vl rt(2, 1);
  for (static int k = 2, s = 2; k < n; k *= 2, s++) {
    rt.resize(n);
    ll z[] = {1, modpow(root, mod >> s)};
    for(int i=k;i<2*k;i++) rt[i] = rt[i / 2] * z[i & 1] % mod;
  }
  vl rev(n);
  for(int i = 0 ; i < n; i ++ ) rev[i] = (rev[i / 2] | (i & 1) <<
L) / 2;
  for(int i = 0 ; i < n; i ++ ) if (i < rev[i]) swap(a[i],
a[rev[i]]);
  for (int k = 1; k < n; k *= 2)
    for (int i = 0; i < n; i += 2 * k) for(int j=0;j<k;j++) {
      ll z = rt[j + k] * a[i + j + k] % mod, &ai = a[i + j];
      a[i + j + k] = ai - z + (z > ai ? mod : 0);
      ai += (ai + z >= mod ? z - mod : z);
    }
}
vl conv(const vl &a, const vl &b) {
  if (a.empty() || b.empty()) return {};
  int s = a.size() + b.size() - 1, B = 32 - __builtin_clz(s),
      n = 1 << B;
  int inv = modpow(n, mod - 2);
  vl L(a), R(b), out(n);
  L.resize(n), R.resize(n);
  ntt(L), ntt(R);
  for(int i = 0 ; i < n;i ++ )
    out[-i & (n - 1)] = (ll)L[i] * R[i] % mod * inv % mod;
  ntt(out);
  return {out.begin(), out.begin() + s};
}
```

11.3. Sum of n^k in $O(k^2)$

```
LL mod;
LL S[105][105];
void solve() {
    LL n, k;
    scanf("%lld %lld %lld", &n, &k, &mod);
    S[0][0] = 1 % mod;
    for (int i = 1; i <= k; i++) {
        for (int j = 1; j <= i; j++) {
            if (i == j) S[i][j] = 1 % mod;
            else S[i][j] = (j * S[i - 1][j] + S[i - 1][j - 1]) %
mod;
        }
    }
    LL ans = 0;
    for (int i = 0; i <= k; i++) {
        LL fact = 1, z = i + 1;
        for (LL j = n - i + 1; j <= n + 1; j++) {
            LL mul = j;
            if (mul % z == 0) {
                mul /= z;
                z /= z;
            }
            fact = (fact * mul) % mod;
        }
        ans = (ans + S[k][i] * fact) % mod;
    }
    printf("%lld\n", ans);
}
```

11.4. Gauss method

```
const double EPS = 1e-9;
const int INF = 2; // it doesn't actually have to be infinity or
a big number

int gauss (vector < vector<double> > a, vector<double> & ans) {
    int n = (int) a.size();
    int m = (int) a[0].size() - 1;

    vector<int> where (m, -1);
    for (int col=0, row=0; col<m && row<n; ++col) {
        int sel = row;
        for (int i=row; i<n; ++i)
            if (abs (a[i][col]) > abs (a[sel][col]))
                sel = i;
        if (abs (a[sel][col]) < EPS)
            continue;
        for (int i=col; i<=m; ++i)
            swap (a[sel][i], a[row][i]);
        where[col] = row;

        for (int i=0; i<n; ++i)
            if (i != row) {
                double c = a[i][col] / a[row][col];
                for (int j=col; j<=m; ++j)
                    a[i][j] -= a[row][j] * c;
            }
        ++row;
    }
    ans.assign (m, 0);
    for (int i=0; i<m; ++i)
        if (where[i] != -1)
            ans[i] = a[where[i]][m] / a[where[i]][i];
    for (int i=0; i<n; ++i) {
        double sum = 0;
        for (int j=0; j<m; ++j)
            sum += ans[j] * a[i][j];
        if (abs (sum - a[i][m]) > EPS)
            return 0;
    }

    for (int i=0; i<m; ++i)
        if (where[i] == -1)
            return INF;
    return 1;
}
```

11.5. Berlekamp-Massey

```
template<typename T>
vector<T> berlekampMassey(const vector<T> &s) {
    vector<T> c;
    vector<T> oldC;
    int f = -1;
    for (int i=0; i<(int)s.size(); i++) {
        T delta = s[i];
        for (int j=1; j<=(int)c.size(); j++)
            delta -= c[j-1] * s[i-j];
        if (delta == 0)
            continue;
        if (f == -1) {

            c.resize(i + 1);
            mt19937 rng(222);
            for (T &x : c)
                x = rng();
            f = i;
        } else {
            vector<T> d = oldC;
            for (T &x : d)
                x = -x;
            d.insert(d.begin(), 1);
            T df1 = 0;  // d(f + 1)
            for (int j=1; j<=(int)d.size(); j++)
                df1 += d[j-1] * s[f+1-j];
            T coef = delta / df1;
            for (T &x : d)
                x *= coef;
            vector<T> zeros(i - f - 1);
            zeros.insert(zeros.end(), d.begin(), d.end());
            d = zeros;
            vector<T> temp = c;
            c.resize(max(c.size(), d.size()));
            for (int j=0; j<(int)d.size(); j++)
                c[j] += d[j];
            if (i - (int) temp.size() > f - (int) oldC.size()) {
                oldC = temp;
                f = i;
```

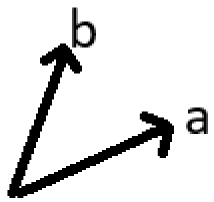
```
            }
        }
    }
    return c;
}
```

12. Our Geometry Template

12.1. Point class

```
template<class T>
struct Point{
    T x;
    T y;
    Point operator+(const Point &o) const {
        return {x + o.x, y + o.y};
    }
    Point operator-(const Point &o) const {
        return {x - o.x, y - o.y};
    }
    Point operator*(T w) const {
        return {x * w, y * w};
    }
    Point operator/(T w) const {
        return {x / w, y / w};
    }
    Point perp() const {
        return Point{-y, x}; // rotates +90 degrees
    }
    bool operator<(Point &o){
        if(x == o.x) return y < o.y;
        else return x < o.x;
    }
    T cross(Point a) const {
        return x * a.y - y * a.x;
    }
    T dist2() const {
        return x * x + y * y;
    }
    double dist() const {
        return sqrt(dist2());
    }
    T operator*(const Point &o) const {
        return x*o.x+y*o.y;
    }
};
```

12.2. Cross Product



In this case $\vec{a} \times \vec{b} = a_x \cdot b_y - a_y \cdot b_x > 0$

12.3. Circumcenter

```
typedef Point<double> P;

double ccRadius(const P& A, const P& B, const P& C) {
    return (B-A).dist()*(C-B).dist()*(A-C).dist()/
    abs((B-A).cross(C-A))/2;
}
P ccCenter(const P& A, const P& B, const P& C) {
    P b = C-A, c = B-A;
    return A + (b*c.dist2()-c*b.dist2()).perp()/b.cross(c)/2;
}
```

12.4. Line Distance

```
typedef Point<double> P;
double lineDist(const P& a, const P& b, const P& p) {
    return (double)(b-a).cross(p-a)/(b-a).dist();
}
```

12.5. Line Intersection

```
typedef Point<double> P;
pair<int, P> lineInter(P s1, P e1, P s2, P e2) {
    auto d = (e1 - s1).cross(e2 - s2);
    if (d == 0) // i f pa ra l l e l
        return {-(s1.cross(e1, s2) == 0), P(0, 0)};
    auto p = s2.cross(e1, e2), q = s2.cross(e2, s1);
    return {1, (s1 * p + e1 * q) / d};
}
```

12.6. Minimum-Enclosing Circle

```
typedef Point<double> P;

pair<P, double> enclose(vector<P> ps) {
    shuffle(ps.begin(), ps.end(), mt19937(time(0)));
    P o = ps[0];
    double r = 0, EPS = 1 + 1e-8;
    int sz = (int)ps.size();
    for(int i = 0 ; i < sz; i ++ ){
        if((o - ps[i]).dist() > r * EPS){
            o = ps[i], r =0;
            for(int j = 0 ; j < i; j ++ ){
                if((o - ps[j]).dist() > r * EPS){
                    o = (ps[i] + ps[j]) / 2;
                    r = (o - ps[i]).dist();
                    for(int k = 0 ; k < j ; k ++ ){
                        if((o - ps[k]).dist() > r * EPS){
                            o = ccCenter(ps[i], ps[j], ps[k]);
                            r = (o - ps[i]).dist();
                        }
                    }
                }
            }
        }
    }
    return {o, r};
}
```

12.7. Polar-Sort

```
sort(X.begin(), X.end(), [&](Point<int> a, Point<int> b){
    Point<int> origin{0, 0};
    bool ba = a < origin, bb = b < origin;
    if(ba != bb) {return ba < bb;}
    else return a.cross(b) > 0;
});
```

13. General

13.1. Simulated Annealing

```
const ld T = (ld)2000;
const ld alpha = 0.999999;
// (new_score - old_score) / (temperature_final) ~ 10 works well

const ld L = (ld)1e6;
ld small_rand(){
  return ((ld)gen(L))/L;
}

ld P(ld old, ld nw, ld temp){
  if(nw > old)
    return 1.0;
  return exp((nw-old)/temp);
}

{
  auto start = chrono::steady_clock::now();
  ld time_limit = 2000;
  ld temperature = T;
  ld max_score = -1;

  while(elapsed_time < time_limit){
    auto cur = chrono::steady_clock::now();
    elapsed_time =
chrono::duration_cast<chrono::milliseconds>(cur - start).count();
    temperature *= alpha;

    // try a neighboring state
    // ....
    // ....

    old_score = score(old_state);
    new_score = score(new_state);
    if(P(old_score, new_score, temperature) >= small_rand()){
      old_state = new_state;
      old_score = new_score;
    }
    if(old_score > max_score){
      max_score = old_score;
      max_state = old_state;
    }
  }
}
```

14. janY's 2D Geometry

14.1. vec2

```
typedef long double ld;

const ld eps = 1e-9, inf = 1e9;

template<typename V>
struct vec2 {
    V x, y;

    vec2(): x(0), y(0) {}
    vec2(V x, V y): x(x), y(y) {}
    // optional conversion constructor
    template<typename U> vec2(const vec2<U>& other) : x(other.x),
y(other.y) {}

    // optional
    V length() {return sqrt(x * x + y * y);}

    V dot(vec2<V> other) {return x*other.x + y*other.y;}

    // a.cross(b)>0: b is to the left of a
    // a.cross(b)<0: b is to the right of a
    // a.cross(b)=0: vectors are collinear (same or opposite
direction)
    V cross(vec2<V> other) {return x*other.y - y*other.x;}

    // optional
    void reduce(bool pos_x = false) { // only for integer
        V g = __gcd(x, y);
        if (g == 0) return;
        if (g < 0) g = -g;
        x /= g;
        y /= g;
        // ensure canonical representation for direction
        if (pos_x && (x < 0 || (x == 0 && y < 0))) {
            x = -x;
            y = -y;
        }
    }

    // optional (need .length())
    void normalize() { // only for floating point
        V len = this->length();
        if (len == 0) return;
        x /= len;
        y /= len;
    }

    // optional, rotate angle radians around (0, 0).
    void rotate(ld angle) { // only for floating point
        ld sin_angle = sin(angle);
        ld cos_angle = cos(angle);
        V new_x = x*cos_angle - y*sin_angle;
        y = x*sin_angle + y*cos_angle;
        x = new_x;
    }

    // optional, returns new vec2 rotated angle radians around
(0, 0).
    vec2 rotated(ld angle) {
        vec2<V> thiz = *this;
        thiz.rotate(angle);
        return thiz;
    }

    // optional
    void rotate90() {V new_x = -y; y = x; x = new_x;}

    // optional, returns angle between two directions, always
positive
    ld angle_to(vec2<V> w) {
        ld cos_theta = (dot(w) / w.length()) / length();
        return acos(max((ld)-1.0, min((ld)1.0, cos_theta)));
    }

    //optional addition/subtraction
    vec2 operator+(const vec2<V>& other) {return vec2(x +
other.x, y + other.y);}
    vec2 operator-(const vec2<V>& other) {return vec2(x -
other.x, y - other.y);}

    // optional scalar multiplication
    vec2 operator*(const V& k) {return vec2(x * k, y * k);}
    vec2& operator*=(const V& k) {x *= k; y *= k; return *this;}

    // optional scalar division
    vec2 operator/(const V& k) {return vec2(x / k, y / k);}
    vec2& operator/=(const V& k) {x /= k; y /= k; return *this;}

    // optional equality operators
    bool operator==(const vec2<V>& other) {return x == other.x &&
y == other.y;}
    bool operator!=(const vec2<V>& other) {return !(*this ==
other);}
};

// optional nice cout
template<typename V>
ostream& operator<<(ostream& os,  const vec2<V>& v) {
    return os << "(" << v.x << " " << v.y << ")";
}

// optional nice cin
template<typename V>
istream& operator>>(istream& is, vec2<V>& v) {
    return is >> v.x >> v.y;
}
```

14.2. 2D Geometric Functions

```
// line line intersection
// returns true if exists, stores result in out
bool line_intersection(vec2<ld> p1, vec2<ld> d1, vec2<ld> p2,
vec2<ld> d2, vec2<ld> &out) {
    ld cross_d = d1.cross(d2);
    if (abs(cross_d) < 1e-10) return false;
    vec2<ld> r = p2 - p1;
    ld t1 = r.cross(d2) / cross_d;
    out = p1 + d1 * t1;
    return true;
}

// circle circle intersection
// returns true if exists, stores result in out
bool circle_circle(vec2<ld> c1, ld r1, vec2<ld> c2, ld r2,
pair<vec2<ld>,vec2<ld>> &out) {
    ld d = (c2-c1).length();
    ld co = (d*d + r1*r1 - r2*r2)/(2*d*r1);
    if (abs(co) > 1) return false;
    ld alpha = acos(co);
    vec2<ld> rad = (c2-c1)/d*r1; // vector C1C2 resized to have
length r1
    out = {c1 + rad.rotated(-alpha), c1 + rad.rotated(alpha)};
    return true;
}

// quadratic formula a*x^2 + b*x + c = 0
// returns root count and stores result in out
int quad_roots(ld a, ld b, ld c, pair<ld,ld> &out) {
    assert(a != 0);
    ld disc = b*b - 4*a*c;
    if (disc < 0) return 0;
    ld sum = (b >= 0) ? -b-sqrt(disc) : -b+sqrt(disc);
    out = {sum/(2*a), sum == 0 ? 0 : (2*c)/sum};
    return 1 + (disc > 0);
}

struct StableSum {
    int cnt = 0;
    vector<ld> v, pref{0};
    void operator+=(ld a) {
        assert(a >= 0);
        int s = ++cnt;
        while (s % 2 == 0) {
            a += v.back();
            v.pop_back(), pref.pop_back();
            s /= 2;
        }
        v.push_back(a);
        pref.push_back(pref.back() + a);
    }
    ld val() {return pref.back();}
};

// sorts starting from (1, 0) inclusive going clockwise.
template<typename T> bool half(const vec2<T> &p) {
    return (p.y < 0 || (p.y == 0 && p.x < 0));
}
template<typename T> void polar_sort(vector<vec2<T>> &v) {
    sort(v.begin(), v.end(), [](vec2<T> v, vec2<T> w) {
        return make_tuple(half(v), 0) < make_tuple(half(w),
v.cross(w));
    });
}
```

14.3. Halfplane Intersection

```
// Basic half-plane struct.
struct Halfplane {
    // 'p' is a passing point of the line and 'pq' is the
direction vector of the line.
```

```
    vec2<ld> p, pq;
    ld angle;

    Halfplane() {}
    Halfplane(vec2<ld> a, vec2<ld> 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(vec2<ld> r) {
        return pq.cross(r - p) < -eps;
    }

    // Comparator for sorting.
    bool operator<(const Halfplane& e) const {
        return angle < e.angle;
    }

    // Intersection point of the lines of two half-planes. It is
assumed they're never parallel.
    vec2<ld> inter(Halfplane& t) {
        ld alpha = (t.p - p).cross(t.pq) / pq.cross(t.pq);
        return p + (pq * alpha);
    }
};

vector<vec2<ld>> hp_intersect(vector<Halfplane>& H) {

    vec2<ld> box[4] = {  // Bounding box in CCW order
        vec2<ld>(inf, inf),
        vec2<ld>(-inf, inf),
        vec2<ld>(-inf, -inf),
        vec2<ld>(inf, -inf)
    };

    for(int i = 0; i<4; i++) { // Add bounding box half-planes.
        Halfplane aux(box[i], box[(i+1) % 4]);
        H.push_back(aux);
    }

    // Sort by angle and start algorithm
    sort(H.begin(), 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(dq[len-1].inter(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(dq[0].inter(dq[1]))) {
            dq.pop_front();
            --len;
        }

        // Special case check: Parallel half-planes
        if (len > 0 && fabsl(H[i].pq.cross(dq[len-1].pq)) < eps)
{
            // Opposite parallel half-planes that ended up
checked against each other.
            if (H[i].pq.dot(dq[len-1].pq) < 0.0)
                return vector<vec2<ld>>();

            // Same direction half-plane: keep only the leftmost
half-plane.
            if (H[i].out(dq[len-1].p)) {
                dq.pop_back();
                --len;
            }
            else continue;
        }

        // 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(dq[len-1].inter(dq[len-2]))) {
        dq.pop_back();
        --len;
    }

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

    // Report empty intersection if necessary
    if (len < 3) return vector<vec2<ld>>();

    // Reconstruct the convex polygon from the remaining half-
planes.
    vector<vec2<ld>> ret(len);
    for(int i = 0; i+1 < len; i++) {
        ret[i] = dq[i].inter(dq[i+1]);
    }
    ret.back() = dq[len-1].inter(dq[0]);

    // Check if area is non-zero
    ld area = 0;
    for(int i = 0; i < ret.size(); i++) {
        int nxt = i+1;
        if (nxt == ret.size()) nxt = 0;
        area += ret[i].x*ret[nxt].y - ret[i].y*ret[nxt].x;
    }
    if (abs(area)/2.0 < eps) return vector<vec2<ld>>();

    return ret;
}
```

15. janY's Algorithms

15.1. Modulo

```
int mod;
int mod_f(long long a){
    return ((a%mod)+mod)%mod;
}

int m_add(long long a, long long b){
    return mod_f((long long)mod_f(a) + mod_f(b));
}

int m_mult(long long a, long long b){
    return mod_f((long long)mod_f(a) * mod_f(b));
}

// C function for extended Euclidean Algorithm (used to
// find modular inverse.
int gcdExt(int a, int b, int *x, int *y) {
    // Base Case
    if (a == 0){
        *x = 0, *y = 1;
        return b;
    }

    int x1, y1; // To store results of recursive call
    int gcd = gcdExt(b%a, a, &x1, &y1);

    // Update x and y using results of recursive
    // call
    *x = y1 - (b/a) * x1;
    *y = x1;
    return gcd;
}

// Function to find modulo inverse of b. It returns
// -1 when inverse doesn't
int modInverse(int b, int m) {
    int x, y; // used in extended GCD algorithm
    int g = gcdExt(b, m, &x, &y);

    // Return -1 if b and m are not co-prime
    if (g != 1) return -1;

    // m is added to handle negative x
    return (x%m + m) % m;
}

// Function to compute a/b under modulo m
int m_divide(long long a, long long b) {
    a = a % mod;
    int inv = modInverse(b, mod);
    if (inv == -1)
       return -1;
    else
       return (inv * a) % mod;
}

// exponent function (with mod)
```

```
int m_pow(long long base, long long exp) {
    base %= mod;
    int result = 1;
    while (exp > 0) {
        if (exp & 1) result = ((long long)result * base) % mod;
        base = ((long long)base * base) % mod;
        exp >>= 1;
    }
    return result;
}
```

15.2. Factorization

```
vector<int> getPrimes(int n)
{
    vector<int> res;
    bool prime[n + 1];
    memset(prime, true, sizeof(prime));
    for (ll p = 2; p * p <= n; p++) {
        if (prime[p] == true) {
            for (ll i = p * p; i <= n; i += p){
                prime[i] = false;
            }
        }
    }
    for (int p = 2; p <= n; p++){
        if (prime[p]){
            res.push_back(p);
        }
    }
    return res;
}

// only prime factors
vector<int> primes;
vector<int> get_prime_factors(long long n){
    vector<int> factors;
    if (n == 1) return factors;
    for (auto &i : primes){
        if (i*i > n) break;
        if (n%i==0){
            factors.push_back(i);
            while (n%i==0) n/=i;
        }
    }
    if (n != 1) factors.push_back(n);
    return factors;
}

map<int,int> get_prime_factors(long long n){
    map<int,int> factors;
    if (n == 1) return factors;
    for (auto &i : primes){
        if (i*i > n) break;
        while (n%i==0) {
            factors[i]++;
            n/=i;
        }
    }
    if (n != 1) factors[n]++;
    return factors;
}

vector<int> get_factors(long long n){
    vector<int> factors;
    for (int i = 1; i*i <= n; i++){
        if (n%i==0){
            factors.push_back(i);
            if (i*i != n) factors.push_back(n/i);
        }
    }
    //factors.push_back(n);
    return factors;
}
```

15.3. Combinatorics

```
long long nPr(long long n, long long r){
    if (r > n) return 0;
    if (n-r < r) r = n-r;
    long long count = r;
    long long result = 1;
    while (count > 0){
        //result = m_mult(result, n);
        result = result * n;
        n--;
        count--;
    }
    return result;
}

// slow nCr
long long nCr(long long n, long long r){
    if (r > n) return 0;
    if (n-r < r) r = n-r;
    long long count = r;
    long long result = 1;
    while (count > 0){
        //result = m_mult(result, n);
        result = result * n;
        n--;
        count--;
    }
    long long num = 1;
    while (num <= r){
        //result = m_divide(result, num);
        result = result / num;
        num++;
    }
    return result;
}

// fast nCr (REQ modulo m_mult, modInverse)
int MAX_CHOOSE = 3e5;
vector<long long> inverse_fact(MAX_CHOOSE+5);
vector<long long> fact(MAX_CHOOSE+5);

long long fast_nCr(long long n, long long r) {
    if (n < r || r < 0) return 0;
    return (((fact[n] * inverse_fact[r]) % mod) * inverse_fact[n-
r]) % mod;
}

void precalc_fact(int n){
    fact[0] = fact[1] = 1;
    for (long long i = 2; i <= n; i++){
        fact[i] = (fact[i-1]*i) % mod;
    }
    inverse_fact[0] = inverse_fact[1] = 1;
    for (long long i = 2; i <= n; i++){
        inverse_fact[i] = (modInverse(i, mod) *
inverse_fact[i-1]) % mod;
    }
}
```

15.4. Disjoint Set Union

```
struct disjSet { // Disjoint set
    int *rank, *parent, n;
    disjSet() {}
  disjSet(int n) {init(n);}
    void init(int n){
        rank = new int[n];
    parent = new int[n];
    this->n = n;
    for (int i = 0; i < n; i++) {
            rank[i] = 0;
            parent[i] = i;
        }
    }
  int find(int a) {
    if (parent[a] != a){
            //return find(parent[a]); // no path compression
            parent[a] = find(parent[a]); // path compression
        }
    return parent[a];
  }
  void Union(int a, int b) {
    int a_set = find(a);
    int b_set = find(b);
    if (a_set == b_set) return;
    if (rank[a_set] < rank[b_set]) {
            update_union(a_set, b_set);
    } else if (rank[a_set] > rank[b_set]) {
            update_union(b_set, a_set);
    } else {
            update_union(b_set, a_set);
      rank[a_set] = rank[a_set] + 1;
    }
  }
    // change merge behaviour here
    void update_union(int a, int b){ // merge a into b
        parent[a] = b;
    }
};
```

15.5. Merge Sort Tree

```
struct MergeSortTree {

    int size;
```

```
    vector<vector<ll>> values;

    void init(int n){
        size = 1;
        while (size < n){
            size *= 2;
        }
        values.resize(size*2, vl(0));
    }

    void build(vl &arr, int x, int lx, int rx){
        if (rx - lx == 1){
            if (lx < arr.size()){
                values[x].pb(arr[lx]);
            } else {
                values[x].pb(-1);
            }
            return;
        }
        int m = (lx+rx)/2;
        build(arr, 2 * x + 1, lx, m);
        build(arr, 2 * x + 2, m, rx);

        int i = 0;
        int j = 0;
        int asize = values[2*x+1].size();
        while (i < asize && j < asize){
            if (values[2*x+1][i] < values[2*x+2][j]){
                values[x].pb(values[2*x+1][i]);
                i++;
            } else {
                values[x].pb(values[2*x+2][j]);
                j++;
            }
        }
        while (i < asize) {
            values[x].pb(values[2*x+1][i]);
            i++;
        }
        while (j < asize){
            values[x].pb(values[2*x+2][j]);
            j++;
        }
    }
    void build(vl &arr){
        build(arr, 0, 0, size);
    }

    int calc(int l, int r, int x, int lx, int rx, int k){
        if (lx >= r || rx <= l) return 0;

        if (lx >= l && rx <= r) { // CHANGE HEURISTIC HERE
(elements strictly less than k currently)
            int lft = -1;
            int rght = values[x].size();
            while (rght - lft > 1){
                int mid = (lft+rght)/2;
                if (values[x][mid] < k){
                    lft = mid;
                } else {
                    rght = mid;
                }
            }
            return lft+1;
        }

        int m = (lx+rx)/2;
        int values1 = calc(l, r, 2*x+1, lx, m, k);
        int values2 = calc(l, r, 2*x+2, m, rx, k);
        return values1 + values2;
    }
    int calc(int l, int r, int k){
        return calc(l, r, 0, 0, size, k);
    }
};
```

15.6. Fenwick Tree

```
struct fenwick { // point update (delta), range sum
    ll* bit;
    int fsize;

    fenwick(){}
    fenwick(int n){ init(n); }
    ~fenwick(){ delete[] bit; }

    void init(int n){
        bit = new ll[n+1];
        fsize = n;
        for (int i = 1; i <= n; i++){
            bit[i] = 0;
        }
    }

    int lsb(int x){ // Least significant bit
        return x&(-x);
    }

    ll query(int v){
        ll sum = 0;
        while (v > 0){
            sum += bit[v];
            v -= lsb(v);
        }
        return sum;
    }

    void add(int v, int delta){
        v++; // because 1 indexed
        while (v <= fsize){
            bit[v] += delta;
            v += lsb(v);
        }
    }

    void build(vector<ll> &inp){
        for (int i = 1; i <= inp.size(); i++){
            bit[i] = inp[i-1];
        }
        for (int i = 1; i <= inp.size(); i++){
            int p = i + lsb(i);
            if (p <= fsize){
                bit[p] += bit[i];
            }
        }
    }

    ll calc(int l, int r){ // sum from l to r inclusive (of the
original array)
        return query(r+1) - query(l);
    }
};
```

15.7. Fenwick Tree (Range Updates)

```
struct fenwick { // range update
    ll* bit1;
    ll* bit2;
    int fsize;

    fenwick(){}
    fenwick(int n){
        init(n);
    }

    ~fenwick(){
        delete bit1;
        delete bit2;
    }

    void init(int n){
        bit1 = new ll[n+1];
        bit2 = new ll[n+1];
        fsize = n;
        for (int i = 1; i <= n; i++){
            bit1[i] = 0;
            bit2[i] = 0;
        }
    }

    ll getSum(ll BITree[], int index){
        ll sum = 0;
        index++;
        while (index > 0) {
            sum += BITree[index];
            index -= index & (-index);
        }
        return sum;
    }

    void updateBIT(ll BITree[], int index, ll val){
        index++;
        while (index <= fsize) {
            BITree[index] += val;
            index += index & (-index);
        }
    }

    ll sum(ll x){
        return (getSum(bit1, x) * x) - getSum(bit2, x);
```

```
    }

    void add(ll l, ll r, ll val){ // add val to range l:r
INCLUSIVE
        updateBIT(bit1, l, val);
        updateBIT(bit1, r + 1, -val);
        updateBIT(bit2, l, val * (l - 1));
        updateBIT(bit2, r + 1, -val * r);
    }

    ll calc(ll l, ll r){ // sum on range l:r INCLUSIVE
        return sum(r) - sum(l - 1);
    }
};
```

15.8. Kosaraju's Algorithm

```
// kosaraju's algorithm - find strongly connected components
(SCC) in a directed graph O(V+E)
// V - vertex count, E - edge count
// SCC - every vertex in a component has a path to every other
vertex in the same component
// add directed edges, run .work(), answer stored in scc
struct kosaraju {
    vector<vector<int>> g, gT, scc;
    stack<int> stck;
    vector<int> visited;
    int siz;

    kosaraju() {}
    kosaraju(int siz){
        init(siz);
    }

    void init(int siz){
        this->siz = siz;
        g.assign(siz, vector<int>(0));
        gT.assign(siz, vector<int>(0));
    }

    void add_edge(int u, int v){ // directed edge from u to v
        g[u].push_back(v);
        gT[v].push_back(u);
    }

    void dfs(int loc){
        if (visited[loc]) return;
        visited[loc] = 1;
        for (auto &i : g[loc]) dfs(i);
        stck.push(loc);
    }

    void dfsT(int loc){
        if (visited[loc]) return;
        visited[loc] = 1;
        scc.back().push_back(loc);
        for (auto &i : gT[loc]) dfsT(i);
    }

    void work(){ // call after adding all the edges to get scc
        scc.clear();
        visited.assign(siz, 0);
        for (int i = 0; i < siz; i++) dfs(i);
        visited.assign(siz, 0);
        while (stck.size()){
            int top = stck.top();
            stck.pop();
            if (visited[top] == 0){
                scc.pb(vi(0));
                dfsT(top);
            }
        }
    }
};
```

15.9. Range Minimum Query

```
typedef int typ;
struct RMQ {
    vector<vector<typ>> arr;
    bool do_min_query = 1; // 0 = max query; 1 = min query

    RMQ(){}
    RMQ(vector<typ> &a){
        build(a);
    }
    RMQ(vector<typ> &a, bool is_min) {
        do_min_query = is_min;
        build(a);
    }

    void build(vector<typ> &a){
        arr.pb(a);
        int len = 2, at = 1;
        while (len <= (int)a.size()){
            arr.pb(vector<typ>(0));
            int pos = 0;
            while (pos+len <= (int)a.size()){
                typ val = max(arr[at-1][pos], arr[at-1]
[pos+len/2]);
                if (do_min_query) val = min(arr[at-1][pos],
arr[at-1][pos+len/2]);
                arr[at].pb(val);
                pos++;
            }
            len *= 2; at++;
        }
    }

    typ calc(int l, int r){ // 0 indexed, [l; r] inclusive
        int dist = r-l+1;
        int bigbit = 31-__builtin_clz(dist);
        typ ret = max(arr[bigbit][l], arr[bigbit][r-
(1<<bigbit)+1]);
        if (do_min_query) ret = min(arr[bigbit][l], arr[bigbit]
[r-(1<<bigbit)+1]);
        return ret;
    }
};
```

15.10. Polynomial Rolling Hash

```
int gcdExt(int a, int b, int *x, int *y) {
    if (a == 0){
        *x = 0, *y = 1;
        return b;
    }
    int x1, y1;
    int gcd = gcdExt(b%a, a, &x1, &y1);
    *x = y1 - (b/a) * x1;
    *y = x1;
    return gcd;
}

int modInverse(int b, int m) {
    int x, y;
    int g = gcdExt(b, m, &x, &y);
    if (g != 1) return -1;
    return (x%m + m) % m;
}

int m_divide(long long a, long long b, long long md) {
    a = a % md;
    int inv = modInverse(b, md);
    if (inv == -1)
       return -1;
    else
       return (inv * a) % md;
}

struct poly_hash {
    const vector<ll> mods = {(ll)1e9+9, (ll)1e9+7};
    const vector<ll> p = {59, 61};
    vector<vector<ll>> pref_hash, powers, divs;

    void work(string &s){
        // precalc powers
        powers.clear();
        powers.resize(2);
        divs.clear();
        divs.resize(2);
        for (int i = 0; i < 2; i++){
            powers[i].push_back(1);
            divs[i].push_back(1);
            for (int j = 0; j < s.size(); j++){
powers[i].push_back((powers[i].back()*p[i])%mods[i]);
                divs[i].push_back(m_divide(divs[i].back(), p[i],
mods[i]));
            }
        }

        pref_hash.clear();
        pref_hash.resize(2);
        for (int i = 0; i < 2; i++){
            pref_hash[i].push_back(0);
            for (int j = 0; j < s.size(); j++){
                int val = s[j]-'a'+1; // a -> 1, b -> 2...
```

```
pref_hash[i].push_back((pref_hash[i].back()+val*powers[i]
[j])%mods[i]);
            }
        }
    }

    pair<ll, ll> calc(int l, int r){ // [l; r] inclusive!
        ll hash1 = ((pref_hash[0][r+1]-pref_hash[0]
[l]+mods[0])*divs[0][l])%mods[0];
        ll hash2 = ((pref_hash[1][r+1]-pref_hash[1]
[l]+mods[1])*divs[1][l])%mods[1];
        return {hash1, hash2};
    }
};
```

15.11. Matrix Template

```
template<typename V> struct Matrix {
    int rows, cols;
    vector<vector<V>> mat;

    Matrix(){}
    Matrix(int row, int col){
        resize(row, col);
    }

    void resize(int row, int col){
        mat.assign(row, vector<V>(col));
        rows = row;
        cols = col;
    }

    Matrix<V> operator* (Matrix<V> const& oth){
        assert(cols == oth.rows);
        Matrix<V> res(rows, oth.cols);
        for (int i = 0; i < rows; i++){
            for (int j = 0; j < oth.cols; j++){
                for (int z = 0; z < cols; z++){
                    res.mat[i][j] += mat[i][z]*oth.mat[z][j];
                    res.mat[i][j] %= mod; // if need mod
                }
            }
        }
        return res;
    }

    Matrix<V> exp(ll pow){ // A^(pow)
        assert(rows == cols);
        Matrix<V> res(rows, cols);
        for (int i = 0; i < rows; i++) res.mat[i][i] = 1;
        Matrix<V> mult = *this;
        while (pow){
            if (pow&1) res = res*mult;
            mult = mult*mult;
            pow /= 2;
        }
        return res;
    }

    void print(){
        int i, j;
        fo(i, rows) {
            fo(j, cols) cout << mat[i][j] << " ";
            cout << "\n";
        }
        cout << "\n";
    }
};
```

15.12. Convex Hull

```
typedef pair<ll, ll> P;
ll cross(P a, P b) {return a.fi*b.se - a.se*b.fi;}
ll cross_from(P start, P a, P b) {
    a.fi -= start.fi;
    a.se -= start.se;
    b.fi -= start.fi;
    b.se -= start.se;
    return cross(a, b);
}
vector<P> convexHull(vector<P> pnts) {
  if (pnts.size() <= 1) return pnts;
  sort(pnts.begin(), pnts.end());
  vector<P> h(pnts.size()+1);
  int s = 0, t = 0;
  for (int it = 2; it--; s = --t, reverse(pnts.begin(),
pnts.end()))
    for (P p : pnts) {
      while (t >= s + 2 && cross_from(h[t-2], h[t-1], p) <= 0)
t--;
      h[t++] = p;
    }
  return {h.begin(), h.begin() + t - (t == 2 && h[0] == h[1])};
}
```

15.13. Prufer Codes

```
// there are n^(n-2) different trees (labeled trees)

// multinomial coefficients: n choose n1, n2, n3, …, nk - number
of ways to partition n elements into k distinct groups of sizes
n1, n2, n3, …, nk.
// n! / (n1! * n2! * … * nk!)

// multinomial theorem:
// (x1 + x2 + … + xm)^p = sum[ci>=0 & sum ci = p] (x1^c1 * x2^c2
* … * xm^cm * (p choose c1, c2, …, ck))

// 0 indexed
vector<vector<int>> prufer_decode(vector<int> &code) {
    vector<vector<int>> g(code.size()+2);
    vector<int> degrees(g.size(), 1);
    for (auto &i : code) degrees[i]++;
    int nxt = g.size(), ptr = 0;
    for (int i = 0; i < code.size(); i++) {
        if (nxt > ptr) {
            while (degrees[ptr] != 1) ptr++;
            nxt = ptr;
        }
        g[nxt].push_back(code[i]);
        g[code[i]].push_back(nxt);
        degrees[nxt] = 0;
        if (--degrees[code[i]] == 1) nxt = code[i];
        else nxt = g.size();
    }
    if (nxt > ptr) while (degrees[ptr] != 1) ptr++;
    else ptr = nxt;
    g[g.size()-1].push_back(ptr);
    g[ptr].push_back(g.size()-1);
    return g;
}

int *prufer_parent;
void dfs(int at, int from, vector<vector<int>> &g) {
    prufer_parent[at] = from;
    for (auto &i : g[at]) {
        if (i == from) continue;
        dfs(i, at, g);
    }
}
// 0 indexed
vector<int> prufer_encode(vector<vector<int>> &g) {
    if (g.size() <= 2) return vector<int>();
    prufer_parent = new int[g.size()];
    dfs(g.size()-1, g.size()-1, g); // will never be removed
    int *degrees = new int[g.size()];
    for (int i = 0; i < g.size(); i++) degrees[i] = g[i].size();
    int nxt = g.size(), ptr = 0;
    vector<int> code(g.size()-2);
    for (int i = 0; i < g.size()-2; i++) {
        if (nxt > ptr) {
            while (degrees[ptr] != 1) ptr++;
            nxt = ptr;
        }
        code[i] = prufer_parent[nxt];
        degrees[nxt] = 0;
        if (--degrees[prufer_parent[nxt]] == 1) nxt =
prufer_parent[nxt];
        else nxt = g.size();
    }
    return code;
}
```

15.14. Segment tree

```
struct get_item { ll sum; };
struct set_item { ll val; };
struct segtree {
    int size;
    get_item *values;
    get_item NEUTRAL_ELEMENT = {0}; // dont forget to use brain
here please

    get_item merge(get_item &a, get_item &b){
        return {a.sum + b.sum};
    }
    void apply(get_item &x, set_item &y){
        x.sum = y.val;
    }

    void init(int n){
```

```
        size = 1;
        while (size < n) size *= 2;
        values = new get_item[size*2];
        fill_n(values, size*2, NEUTRAL_ELEMENT);
    }

    void upd(int i, set_item &v, int x, int lx, int rx){
        if (rx - lx == 1){
            apply(values[x], v);
            return;
        }
        int m = (lx + rx) / 2;
        if (i < m) upd(i, v, 2*x+1, lx, m);
        else upd(i, v, 2*x+2, m, rx);
        values[x] = merge(values[2*x+1], values[2*x+2]);
    }
    void upd(int i, set_item v){
        upd(i, v, 0, 0, size);
    }

    get_item calc(int l, int r, int x, int lx, int rx){
        if (lx >= r || rx <= l) return NEUTRAL_ELEMENT;
        if (lx >= l && rx <= r) return values[x];
        int m = (lx+rx)/2;
        get_item v1 = calc(l, r, 2*x+1, lx, m);
        get_item v2 = calc(l, r, 2*x+2, m, rx);
        return merge(v1, v2);
    }
    // INCLUSIVE
    get_item calc(int l, int r){
        return calc(l, r+1, 0, 0, size);
    }

    void build(vector<set_item> &arr, int x, int lx, int rx){ //
optional
        if (rx - lx == 1){
            if (lx < arr.size()) apply(values[x], arr[lx]);
            return;
        }
        int m = (lx+rx)/2;
        build(arr, 2*x+1, lx, m);
        build(arr, 2*x+2, m, rx);
        values[x] = merge(values[2*x+1], values[2*x+2]);
    }
    void build(vector<set_item> &arr){
        build(arr, 0, 0, size);
    }
};
```

15.15. NTT

```
int gcdExt(int a, int b, int *x, int *y) {
    // Base Case
    if (a == 0){
        *x = 0, *y = 1;
        return b;
    }

    int x1, y1; // To store results of recursive call
    int gcd = gcdExt(b%a, a, &x1, &y1);

    // Update x and y using results of recursive
    // call
    *x = y1 - (b/a) * x1;
    *y = x1;
    return gcd;
}

int modInverse(int b, int m) {
    int x, y;
    int g = gcdExt(b, m, &x, &y);
    if (g != 1) return -1;
    return (x%m + m) % m;
}

const int mod = 998244353;
const int root = 15311432;
const int root_1 = modInverse(root, mod);
const int root_pw = 1 << 23;

void fft(vector<int> & a, bool invert) {
    int n = a.size();

    for (int i = 1, j = 0; i < n; i++) {
        int bit = n >> 1;
        for (; j & bit; bit >>= 1)
            j ^= bit;
        j ^= bit;

        if (i < j)
            swap(a[i], a[j]);
    }

    for (int len = 2; len <= n; len <<= 1) {
        int wlen = invert ? root_1 : root;

        for (int i = len; i < root_pw; i <<= 1)
            wlen = (int)(1LL * wlen * wlen % mod);

        for (int i = 0; i < n; i += len) {
            int w = 1;
            for (int j = 0; j < len / 2; j++) {
                int u = a[i+j], v = (int)(1LL * a[i+j+len/2] * w
% mod);
                a[i+j] = u + v < mod ? u + v : u + v - mod;
                a[i+j+len/2] = u - v >= 0 ? u - v : u - v + mod;
                w = (int)(1LL * w * wlen % mod);
            }
        }
    }

    if (invert) {
        int n_1 = modInverse(n, mod);
        for (int & x : a)
            x = (int)(1LL * x * n_1 % mod);
    }
}

// polynomial multiplication
vector<int> multiply(vector<int> const& a, vector<int> const& b)
{
    vector<int> fa(a.begin(), a.end()), fb(b.begin(), b.end());
    int n = 1;
    while (n < a.size() + b.size())
        n <<= 1;
    fa.resize(n);
    fb.resize(n);

    fft(fa, false);
    if (a == b) {
        fb = fa;
    } else {
        fft(fb, false);
    }

    for (int i = 0; i < n; i++) {
        fa[i] = ((ll)fa[i]*fb[i])%mod;
    }
    fft(fa, true);

    vector<int> result(a.size() + b.size() - 1);
    for (int i = 0; i < result.size(); i++)
        result[i] = fa[i];
    return result;
}

vi powz(vi a, ll p){
    vi base = a;
    vi ans(1, 1);
    while (p) {
        if (p&1) ans = multiply(ans, base);
        base = multiply(base, base);
        p >>= 1;
    }
    return ans;
}
```

16. Out of ideas?

1. $\mathrm{opt}(i) \leq \mathrm{opt}(i+1)$
2. Divide & Conquer on queries. Object A_i exists for query range $[L_i; R_i]$ - construct a segtree assuming A_i can be rerolled.