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: $a^{\varphi(n)} \equiv 1 (\mathrm{mod}\, n)$ 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
}
```

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

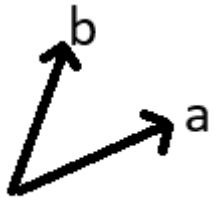
```
const double EPS = 1e-9;
const int INF = 2; // it doesn't actually have to be infinity or
a big number

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;
}
```

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. Out of ideas?

1. $\text{opt}(i) \leq \text{opt}(i+1)$