NTJ
UDESC

Eric Grochowicz, Enzo de Almeida Rodrigues e João Marcos de Oliveira

15 de outubro de 2023

Índice

1 Strings **3**

1.1 Automato de Aho Corasick . . . 3

1.2 Hashing estatico e dinamico . . . 4

1.3 KMP . . . 8

1.4 Suffix Automaton . . . 9

2 Problemas **12**

2.1 Kth digito na string infinita de digitos . . . 12

3 Estruturas **13**

3.1 Fenwick Tree . . . 13

3.2 Segment Tree Beats . . . 13

4 Grafos **18**

4.1 Binary Lifting . . . 18

4.2 Binary Lifting Query (em arestas) . . . 19

4.3 Binary Lifting Query (em nodos) . . . 20

4.4 Binary Lifting Query (em nodos) . . . 21

4.5 Bridges e Edge Biconnected Components . . . 23

4.6 Dinic . . . 24

4.7 Pontos de articulacao . . . 26

5 Matematica **28**

5.1 Crivo de Eratostenes . . . 28

5.2 Fast Fourier Transform . . . 28

5.3 Pollard Rho . . . 30

6 Geometria **32**

6.1 Geometria inteiro 32

7 Extra **34**

7.1 Config do Vim 34

7.2 Custom Hash 34

7.3 Gerador aleatorio de casos 34

7.4 Mint easier 35

7.5 Mint faster 35

7.6 Rand C++ 36

7.7 Script de stress test 36

7.8 Script pra rodar C++ 36

7.9 Template C++ 36

7.10 Template de debug simples 36

1 Strings

1.1 Automato de Aho Corasick

```
// Fonte: https://github.com/shahjalalshohag/code-library
//
// Faz coisarada

 const int N = 3e5 + 5;

 struct AC {
     int N, P;
     const int A = 26;
     vector<vector<int>> next;
     vector<int> link, out_link;
     vector<vector<int>> out;
     AC() : N(0), P(0) { node(); }
     int node() {
         next.emplace_back(A, 0);
         link.emplace_back(0);
         out_link.emplace_back(0);
         out.emplace_back(0);
         return N++;
     }
     inline int get(char c) { return c - 'a'; }
     int add_pattern(const string T) {
         int u = 0;
         for (auto c : T) {
             if (!next[u][get(c)]) next[u][get(c)] = node();
             u = next[u][get(c)];
         }
         out[u].push_back(P);
         return P++;
     }
     void compute() {
         queue<int> q;
         for (q.push(0); !q.empty();) {
             int u = q.front();
             q.pop();
             for (int c = 0; c < A; ++c) {
                 int v = next[u][c];
                 if (!v)
                     next[u][c] = next[link[u]][c];
                 else {
                     link[v] = u ? next[link[u]][c] : 0;
                     out_link[v] = out[link[v]].empty() ? out_link[link[v]] :
                        link[v];
                     q.push(v);
                 }
             }
         }
     }
     int advance(int u, char c) {
         while (u && !next[u][get(c)]) u = link[u];
         u = next[u][get(c)];
         return u;
     }
 };

 /*
```

```
int32_t main() {
    ios_base::sync_with_stdio(0);
    cin.tie(0);
    auto st = clock();
    int t, cs = 0;
    cin >> t;
    while (t--) {
        int n;
        cin >> n;
        vector<string> v;
        for (int i = 0; i < n; i++) {
            string s;
            cin >> s;
            v.push_back(s);
        }
        sort(v.begin(), v.end());
        v.erase(unique(v.begin(), v.end()), v.end());
        AC aho;
        vector<int> len(n + 3, 0);
        for (auto s : v) {
            len[aho.add_pattern(s)] = s.size();
        }
        aho.compute();
        string s;
        cin >> s;
        n = s.size();
        vector<int> dp(n, n + 10);
        int u = 0;
        for (int i = 0; i < n; i++) {
            char c = s[i];
            u = aho.advance(u, c);
            for (int v = u; v; v = aho.out_link[v]) {
                for (auto p : aho.out[v]) {
                    dp[i] = min(dp[i], (i - len[p] >= 0 ? dp[i - len[p]] : 0)
                        + 1);
                }
            }
        }
        cout << "Case " << ++cs << ": ";
        if (dp[n - 1] == n + 10) {
            cout << "impossible\n";
        } else {
            cout << dp[n - 1] << '\n';
        }
    }
    cout << 1.0 * (clock() - st) / 1000 << '\n';
    return 0;
}
*/
```

1.2 Hashing estatico e dinamico

```
// Hashing estatico
// Tambem funciona com vector
//
// Build: O(n)
// Query (operator): O(1)

// Hash com 1 mod
```

```
typedef ll Hash;
template <typename obj = string, typename T = ll, typename U = __int128_t>
   struct Hashing {
    int N;
    bool inverse = 0;
    U p = 1001003;
    T mod = (T)1e18 + 9;
    vector<U> pw, hsh;
    void build(obj s, bool _inverse = 0) {
        inverse = _inverse;
        if (inverse) reverse(begin(s), end(s));
        N = size(s);
        pw.assign(N, 1), hsh.assign(N, s[0] % mod);
        for (int i = 1; i < N; i++) {
            pw[i] = pw[i - 1] * p % mod;
            hsh[i] = (hsh[i - 1] * p + s[i]) % mod;
        }
    }
    Hash operator()(int l, int r) {
        if (inverse) {
            l = N - 1 - l, r = N - 1 - r;
            swap(l, r);
        }
        Hash ans = hsh[r];
        if (l > 0) ans = (ans - (hsh[l - 1] * pw[r - l + 1] % mod) + mod) %
           mod;
        return ans;
    }
};

// Hash com multiplos mods

const int mods = 2;
typedef array<int, mods> Hash; // array<T, mods>

template <typename obj = string, typename T = int, typename U = ll> struct
   Hashing {
    int N;
    bool inverse = 0;
    U p = 1001003;
    Hash mod {998244353, (T)1e9 + 9};
    array<vector<U>, mods> pw, hsh;
    void build(obj s, bool _inverse = 0) {
        inverse = _inverse;
        if (inverse) reverse(begin(s), end(s));
        N = size(s);
        for (int j = 0; j < mods; j++) {
            pw[j].assign(N, 1), hsh[j].assign(N, s[0] % mod[j]);
            for (int i = 1; i < N; i++) {
                pw[j][i] = pw[j][i - 1] * p % mod[j];
                hsh[j][i] = (hsh[j][i - 1] * p + s[i]) % mod[j];
            }
        }
    }
    Hash operator()(int l, int r) {
        if (inverse) {
            l = N - 1 - l, r = N - 1 - r;
            swap(l, r);
        }
        Hash ans;
```

```
        for (int j = 0; j < mods; j++) {
            ans[j] = hsh[j][r];
            if (l > 0) {
                ans[j] = (ans[j] - (hsh[j][l - 1] * pw[j][r - l + 1] %
                    mod[j]) + mod[j]) % mod[j];
            }
        }
        return ans;
    }
};

// Hashing dinamico
//
// Hash que usa uma Fenwick pra updatar
//
// Build: O(n)
// Query (operator): O(log(n))
// Update: O(log(n))

// Hash dinamico com 1 mod

typedef ll Hash;
template <typename obj = string, typename T = ll, typename U = __int128_t>
   struct Hashing {
    int N;
    bool inverse = 0;
    U p = 1001003;
    T mod = (T)1e18 + 9;
    vector<U> pw, hsh, inv;
    void build(obj s, bool _inverse = 0) {
        inverse = _inverse;
        if (inverse) reverse(begin(s), end(s));
        N = size(s);
        pw.resize(N + 1), hsh.resize(N + 1), inv.resize(N + 1);
        pw[1] = p;
        pw[0] = inv[0] = inv[1] = 1;
        U b = p;
        for (T e = mod - 2; e > 0; e >>= 1, b = b * b % mod) {
            if (e & 1) inv[1] = inv[1] * b % mod;
        }
        for (int i = 2; i <= N; i++) {
            inv[i] = inv[i - 1] * inv[1] % mod;
            pw[i] = pw[i - 1] * pw[1] % mod;
        }
        hsh[1] = s[0] * pw[1] % mod;
        for (int i = 2; i <= N; i++) {
            hsh[i] = s[i - 1] * pw[i] % mod;
        }
        for (int i = 1; i <= N; i++) {
            int u = i + (i & -i);
            if (u <= N) hsh[u] = (hsh[u] + hsh[i]) % mod;
        }
    }
    Hash operator()(int l, int r) {
        if (inverse) {
            l = N - 1 - l, r = N - 1 - r;
            swap(l, r);
        }
        l += 1, r += 1;
        Hash ans = 0;
        for (int i = r; i > 0; i -= i & -i) {
```

```
            ans = (ans + hsh[i]) % mod;
        }
        for (int i = l - 1; i > 0; i -= i & -i) {
            ans = (ans - hsh[i] + mod) % mod;
        }
        ans = ans * inv[l - 1] % mod;
        return ans;
    }
    void update(int idx, T val) { // update de soma
        if (inverse) idx = N - 1 - idx;
        idx += 1;
        for (int i = idx; i <= N; i += i & -i) {
            hsh[i] = (hsh[i] + val * pw[idx] % mod) % mod;
        }
    }
};

// Hash dinamico com multiplos mods

const int mods = 2;
typedef array<int, mods> Hash; // array<T, mods>

template <typename obj = string, typename T = int, typename U = ll> struct
   Hashing {
    int N;
    bool inverse = 0;
    U p = 1001003;
    Hash mod {998244353, (T)1e9 + 9};
    array<vector<U>, mods> pw, hsh, inv;
    void build(obj s, bool _inverse = 0) {
        inverse = _inverse;
        if (inverse) reverse(begin(s), end(s));
        N = size(s);
        for (int j = 0; j < mods; j++) {
            pw[j].resize(N + 1), hsh[j].resize(N + 1), inv[j].resize(N + 1);
            pw[j][1] = p;
            pw[j][0] = inv[j][0] = inv[j][1] = 1;
            U b = p;
            for (T e = mod[j] - 2; e > 0; e >>= 1, b = b * b % mod[j]) {
                if (e & 1) inv[j][1] = inv[j][1] * b % mod[j];
            }
            for (int i = 2; i <= N; i++) {
                inv[j][i] = inv[j][i - 1] * inv[j][1] % mod[j];
                pw[j][i] = pw[j][i - 1] * pw[j][1] % mod[j];
            }
            hsh[j][1] = s[0] * pw[j][1] % mod[j];
            for (int i = 2; i <= N; i++) {
                hsh[j][i] = s[i - 1] * pw[j][i] % mod[j];
            }
            for (int i = 1; i <= N; i++) {
                int u = i + (i & -i);
                if (u <= N) hsh[j][u] = (hsh[j][u] + hsh[j][i]) % mod[j];
            }
        }
    }
    Hash operator()(int l, int r) {
        if (inverse) {
            l = N - 1 - l, r = N - 1 - r;
            swap(l, r);
        }
        Hash ans;
```

```
        l += 1, r += 1;
        for (int j = 0; j < mods; j++) {
            ans[j] = 0;
            for (int i = r; i > 0; i -= i & -i) {
                ans[j] = (ans[j] + hsh[j][i]) % mod[j];
            }
            for (int i = l - 1; i > 0; i -= i & -i) {
                ans[j] = (ans[j] - hsh[j][i] + mod[j]) % mod[j];
            }
            ans[j] = ans[j] * inv[j][l - 1] % mod[j];
        }
        return ans;
    }
    void update(int idx, T val) { // update de soma
        if (inverse) idx = N - 1 - idx;
        idx += 1;
        for (int j = 0; j < mods; j++) {
            for (int i = idx; i <= N; i += i & -i) {
                hsh[j][i] = (hsh[j][i] + val * pw[j][idx] % mod[j]) % mod[j];
            }
        }
    }
};
```

1.3 KMP

```
// Fonte: https://github.com/shahjalalshohag/code-library
//
// String matching

const int N = 3e5 + 9;

// returns the longest proper prefix array of pattern p
// where lps[i]=longest proper prefix which is also suffix of p[0...i]
vector<int> build_lps(string p) {
    int sz = p.size();
    vector<int> lps;
    lps.assign(sz + 1, 0);
    int j = 0;
    lps[0] = 0;
    for (int i = 1; i < sz; i++) {
        while (j >= 0 && p[i] != p[j]) {
            if (j >= 1)
                j = lps[j - 1];
            else
                j = -1;
        }
        j++;
        lps[i] = j;
    }
    return lps;
}
vector<int> ans;
// returns matches in vector ans in 0-indexed
void kmp(vector<int> lps, string s, string p) {
    int psz = p.size(), sz = s.size();
    int j = 0;
    for (int i = 0; i < sz; i++) {
        while (j >= 0 && p[j] != s[i])
```

```
            if (j >= 1)
                j = lps[j - 1];
            else
                j = -1;
        j++;
        if (j == psz) {
            j = lps[j - 1];
            // pattern found in string s at position i-psz+1
            ans.push_back(i - psz + 1);
        }
        // after each loop we have j=longest common suffix of s[0..i] which is
        // also prefix of p
    }
}

/*
int main() {
    int i, j, k, n, m, t;
    cin >> t;
    while (t--) {
        string s, p;
        cin >> s >> p;
        vector<int> lps = build_lps(p);
        kmp(lps, s, p);
        if (ans.empty())
            cout << "Not Found\n";
        else {
            cout << ans.size() << endl;
            for (auto x : ans) cout << x << ' ';
            cout << endl;
        }
        ans.clear();
        cout << endl;
    }
    return 0;
}
*/
```

1.4 Suffix Automaton

```
// Fonte: https://github.com/shahjalalshohag/code-library
//
// Faz coisarada

const int N = 3e5 + 9;

// len -> largest string length of the corresponding endpos-equivalent class
// link -> longest suffix that is another endpos-equivalent class.
// firstpos -> 1 indexed end position of the first occurrence of the largest
// string of that node minlen(v) -> smallest string of node v = len(link(v))
   + 1
// terminal nodes -> store the suffixes
struct SuffixAutomaton {
    struct node {
        int len, link, firstpos;
        map<char, int> nxt;
    };
    int sz, last;
    vector<node> t;
```

```
vector<int> terminal;
vector<long long> dp;
vector<vector<int>> g;
SuffixAutomaton() {}
SuffixAutomaton(int n) {
    t.resize(2 * n);
    terminal.resize(2 * n, 0);
    dp.resize(2 * n, -1);
    sz = 1;
    last = 0;
    g.resize(2 * n);
    t[0].len = 0;
    t[0].link = -1;
    t[0].firstpos = 0;
}
void extend(char c) {
    int p = last;
    if (t[p].nxt.count(c)) {
        int q = t[p].nxt[c];
        if (t[q].len == t[p].len + 1) {
            last = q;
            return;
        }
        int clone = sz++;
        t[clone] = t[q];
        t[clone].len = t[p].len + 1;
        t[q].link = clone;
        last = clone;
        while (p != -1 && t[p].nxt[c] == q) {
            t[p].nxt[c] = clone;
            p = t[p].link;
        }
        return;
    }
    int cur = sz++;
    t[cur].len = t[last].len + 1;
    t[cur].firstpos = t[cur].len;
    p = last;
    while (p != -1 && !t[p].nxt.count(c)) {
        t[p].nxt[c] = cur;
        p = t[p].link;
    }
    if (p == -1)
        t[cur].link = 0;
    else {
        int q = t[p].nxt[c];
        if (t[p].len + 1 == t[q].len)
            t[cur].link = q;
        else {
            int clone = sz++;
            t[clone] = t[q];
            t[clone].len = t[p].len + 1;
            while (p != -1 && t[p].nxt[c] == q) {
                t[p].nxt[c] = clone;
                p = t[p].link;
            }
            t[q].link = t[cur].link = clone;
        }
    }
    last = cur;
}
```

```
    void build_tree() {
        for (int i = 1; i < sz; i++) g[t[i].link].push_back(i);
    }
    void build(string &s) {
        for (auto x : s) {
            extend(x);
            terminal[last] = 1;
        }
        build_tree();
    }
    long long cnt(int i) { // number of times i-th node occurs in the string
        if (dp[i] != -1) return dp[i];
        long long ret = terminal[i];
        for (auto &x : g[i]) ret += cnt(x);
        return dp[i] = ret;
    }
};

/*
int32_t main() {
    ios_base::sync_with_stdio(0);
    cin.tie(0);
    int t;
    cin >> t;
    while (t--) {
        string s;
        cin >> s;
        int n = s.size();
        SuffixAutomaton sa(n);
        sa.build(s);
        long long ans = 0; // number of unique substrings
        for (int i = 1; i < sa.sz; i++) ans += sa.t[i].len -
           sa.t[sa.t[i].link].len;
        cout << ans << '\n';
    }
    return 0;
}
*/
```

2 Problemas

2.1 Kth digito na string infinita de digitos

```
// Retorna qual o numero e qual o algarismo do Kth digito
// na string infinita dos numeros naturais (12345678910111213...)
// Complexidade: O(log_10(k))

pair<ll, ll> kthdig(ll k) {
    ll qtd = 1, num_alg = 1, base = 1;
    while (1) {
        ll add = (9 * base) * num_alg;
        if (qtd + add < k) {
            qtd += add;
        } else
            break;
        base *= 10, num_alg++;
    }
    ll algarismo = (k - qtd) % num_alg;
    ll numero = (k - qtd) / num_alg + base;
    return {numero, algarismo};
}
```

3 Estruturas

3.1 Fenwick Tree

```
// Processas queries de operacao com inverso
//
// Build: O(n)
// Query: O(log(n))
// Update: O(log(n))

typedef long long ll;

struct fenwick {
    vector<ll> bit;
    fenwick(int n) { bit.assign(n + 1, 0); }
    fenwick(vector<ll> &v) {
        int n = v.size();
        bit.assign(n + 1, 0);
        for (int i = 1; i <= n; i++) bit[i] = v[i - 1];
        for (int i = 1; i <= n; i++) {
            int j = i + (i & -i);
            if (j <= n) bit[j] += bit[i];
        }
    }
    ll query(int i) {
        ll res = 0;
        for (; i; i -= (i & -i)) res += bit[i];
        return res;
    }
    ll query(int l, int r) { return query(r) - query(l - 1); }
    void update(int i, ll d) {
        for (; i && i < (int)bit.size(); i += (i & -i)) bit[i] += d;
    }
};
```

3.2 Segment Tree Beats

```
// Faz coisarada

const int MAXN = 200001; // 1-based

int N;
ll A[MAXN];

struct Node {
    ll sum;  // Sum tag
    ll max1; // Max value
    ll max2; // Second Max value
    ll maxc; // Max value count
    ll min1; // Min value
    ll min2; // Second Min value
    ll minc; // Min value count
    ll lazy; // Lazy tag
} T[MAXN * 4];

void merge(int t) {
    // sum
    T[t].sum = T[t << 1].sum + T[t << 1 | 1].sum;
```

```
    // max
    if (T[t << 1].max1 == T[t << 1 | 1].max1) {
        T[t].max1 = T[t << 1].max1;
        T[t].max2 = max(T[t << 1].max2, T[t << 1 | 1].max2);
        T[t].maxc = T[t << 1].maxc + T[t << 1 | 1].maxc;
    } else {
        if (T[t << 1].max1 > T[t << 1 | 1].max1) {
            T[t].max1 = T[t << 1].max1;
            T[t].max2 = max(T[t << 1].max2, T[t << 1 | 1].max1);
            T[t].maxc = T[t << 1].maxc;
        } else {
            T[t].max1 = T[t << 1 | 1].max1;
            T[t].max2 = max(T[t << 1].max1, T[t << 1 | 1].max2);
            T[t].maxc = T[t << 1 | 1].maxc;
        }
    }

    // min
    if (T[t << 1].min1 == T[t << 1 | 1].min1) {
        T[t].min1 = T[t << 1].min1;
        T[t].min2 = min(T[t << 1].min2, T[t << 1 | 1].min2);
        T[t].minc = T[t << 1].minc + T[t << 1 | 1].minc;
    } else {
        if (T[t << 1].min1 < T[t << 1 | 1].min1) {
            T[t].min1 = T[t << 1].min1;
            T[t].min2 = min(T[t << 1].min2, T[t << 1 | 1].min1);
            T[t].minc = T[t << 1].minc;
        } else {
            T[t].min1 = T[t << 1 | 1].min1;
            T[t].min2 = min(T[t << 1].min1, T[t << 1 | 1].min2);
            T[t].minc = T[t << 1 | 1].minc;
        }
    }
}

void push_add(int t, int tl, int tr, ll v) {
    if (v == 0) {
        return;
    }
    T[t].sum += (tr - tl + 1) * v;
    T[t].max1 += v;
    if (T[t].max2 != -llINF) {
        T[t].max2 += v;
    }
    T[t].min1 += v;
    if (T[t].min2 != llINF) {
        T[t].min2 += v;
    }
    T[t].lazy += v;
}

// corresponds to a chmin update
void push_max(int t, ll v, bool l) {
    if (v >= T[t].max1) {
        return;
    }
    T[t].sum -= T[t].max1 * T[t].maxc;
    T[t].max1 = v;
    T[t].sum += T[t].max1 * T[t].maxc;
    if (l) {
        T[t].min1 = T[t].max1;
```

```
    } else {
        if (v <= T[t].min1) {
            T[t].min1 = v;
        } else if (v < T[t].min2) {
            T[t].min2 = v;
        }
    }
}

// corresponds to a chmax update
void push_min(int t, ll v, bool l) {
    if (v <= T[t].min1) {
        return;
    }
    T[t].sum -= T[t].min1 * T[t].minc;
    T[t].min1 = v;
    T[t].sum += T[t].min1 * T[t].minc;
    if (l) {
        T[t].max1 = T[t].min1;
    } else {
        if (v >= T[t].max1) {
            T[t].max1 = v;
        } else if (v > T[t].max2) {
            T[t].max2 = v;
        }
    }
}

void pushdown(int t, int tl, int tr) {
    if (tl == tr) return;
    // sum
    int tm = (tl + tr) >> 1;
    push_add(t << 1, tl, tm, T[t].lazy);
    push_add(t << 1 | 1, tm + 1, tr, T[t].lazy);
    T[t].lazy = 0;

    // max
    push_max(t << 1, T[t].max1, tl == tm);
    push_max(t << 1 | 1, T[t].max1, tm + 1 == tr);

    // min
    push_min(t << 1, T[t].min1, tl == tm);
    push_min(t << 1 | 1, T[t].min1, tm + 1 == tr);
}

void build(int t = 1, int tl = 0, int tr = N - 1) {
    T[t].lazy = 0;
    if (tl == tr) {
        T[t].sum = T[t].max1 = T[t].min1 = A[tl];
        T[t].maxc = T[t].minc = 1;
        T[t].max2 = -llINF;
        T[t].min2 = llINF;
        return;
    }

    int tm = (tl + tr) >> 1;
    build(t << 1, tl, tm);
    build(t << 1 | 1, tm + 1, tr);
    merge(t);
}
```

```
void update_add(int l, int r, ll v, int t = 1, int tl = 0, int tr = N - 1) {
    if (r < tl || tr < l) {
        return;
    }
    if (l <= tl && tr <= r) {
        push_add(t, tl, tr, v);
        return;
    }
    pushdown(t, tl, tr);

    int tm = (tl + tr) >> 1;
    update_add(l, r, v, t << 1, tl, tm);
    update_add(l, r, v, t << 1 | 1, tm + 1, tr);
    merge(t);
}

void update_chmin(int l, int r, ll v, int t = 1, int tl = 0, int tr = N - 1) {
    if (r < tl || tr < l || v >= T[t].max1) {
        return;
    }
    if (l <= tl && tr <= r && v > T[t].max2) {
        push_max(t, v, tl == tr);
        return;
    }
    pushdown(t, tl, tr);

    int tm = (tl + tr) >> 1;
    update_chmin(l, r, v, t << 1, tl, tm);
    update_chmin(l, r, v, t << 1 | 1, tm + 1, tr);
    merge(t);
}

void update_chmax(int l, int r, ll v, int t = 1, int tl = 0, int tr = N - 1) {
    if (r < tl || tr < l || v <= T[t].min1) {
        return;
    }
    if (l <= tl && tr <= r && v < T[t].min2) {
        push_min(t, v, tl == tr);
        return;
    }
    pushdown(t, tl, tr);

    int tm = (tl + tr) >> 1;
    update_chmax(l, r, v, t << 1, tl, tm);
    update_chmax(l, r, v, t << 1 | 1, tm + 1, tr);
    merge(t);
}

ll query_sum(int l, int r, int t = 1, int tl = 0, int tr = N - 1) {
    if (r < tl || tr < l) {
        return 0;
    }
    if (l <= tl && tr <= r) {
        return T[t].sum;
    }
    pushdown(t, tl, tr);

    int tm = (tl + tr) >> 1;
    return query_sum(l, r, t << 1, tl, tm) + query_sum(l, r, t << 1 | 1, tm +
        1, tr);
}
```

```
/*
int main() {
    int Q;

    cin >> N >> Q;
    for (int i = 0; i < N; i++) {
        cin >> A[i];
    }
    build();
    for (int q = 0; q < Q; q++) {
        int t;
        cin >> t;
        if (t == 0) {
            int l, r;
            ll x;
            cin >> l >> r >> x;
            update_chmin(l, r - 1, x);
        } else if (t == 1) {
            int l, r;
            ll x;
            cin >> l >> r >> x;
            update_chmax(l, r - 1, x);
        } else if (t == 2) {
            int l, r;
            ll x;
            cin >> l >> r >> x;
            update_add(l, r - 1, x);
        } else if (t == 3) {
            int l, r;
            cin >> l >> r;
            cout << query_sum(l, r - 1) << '\n';
        }
    }
}
*/
```

4 Grafos

4.1 Binary Lifting

```
// Binary Lifting pra LCA
//
// Computa Lowest Common Ancestor e faz queries de k-esimo ancestral
//
// Build(): O(n log(n))
// Lca(): O(log(n))
// Kth(): O(log(n))
//
// up[u][i] = (2 ^ i)-esimo pai do u

 struct BinaryLifting {
     vector<vector<int>> adj, up;
     vector<int> tin, tout;
     int N, LG, t;

     const int neutral = 0;
     int merge(int l, int r) { return l + r; }

     void dfs(int u, int p = -1) {
         tin[u] = t++;
         for (int i = 0; i < LG - 1; i++) up[u][i + 1] = up[up[u][i]][i];
         for (int v : adj[u]) if (v != p) {
             up[v][0] = u;
             dfs(v, u);
         }
         tout[u] = t++;
     }

     void build(int root, vector<vector<int>>& adj2) {
         t = 1;
         N = size(adj2);
         LG = 31 - __builtin_clz(N);
         adj = adj2;
         tin = tout = vector<int>(N);
         up = vector (N, vector<int>(LG));
         up[root][0] = root;
         dfs(root);
     }

     bool ancestor(int u, int v) { return tin[u] <= tin[v] && tout[u] >=
        tout[v]; }

     int lca(int u, int v) {
         if (ancestor(u, v)) return u;
         if (ancestor(v, u)) return v;
         for (int i = LG - 1; i >= 0; i--) {
             if (!ancestor(up[u][i], v)) u = up[u][i];
         }
         return up[u][0];
     }

     int kth(int u, int k) {
         for (int i = 0; i < LG; i++) {
             if (k & (1 << i)) u = up[u][i];
         }
         return u;
```

```
    }

} bl;
```

4.2 Binary Lifting Query (em arestas)

```
// Resolve queries em arvore quando os valores
// estao nas arestas
//
// Build(): O(n log(n))
// query(): O(log(n))
//
// up[u][i] = (2 ^ i)-esimo pai do u
// st[u][i] = query ate (2 ^ i)-esimo pai do u

struct BinaryLifting {
    vector<vector<ii>> adj;
    vector<vector<int>> up, st;
    vector<int> tin, tout;
    int N, LG, t;

    const int neutral = 0;
    int merge(int l, int r) { return l + r; }

    void dfs(int u, int p = -1) {
        tin[u] = t++;
        for (int i = 0; i < LG - 1; i++) {
            up[u][i + 1] = up[up[u][i]][i];
            st[u][i + 1] = merge(st[u][i], st[up[u][i]][i]);
        }
        for (auto [w, v] : adj[u])
            if (v != p) {
                up[v][0] = u, st[v][0] = w;
                dfs(v, u);
            }
        tout[u] = t++;
    }

    void build(int root, vector<vector<ii>>& adj2) {
        t = 1;
        N = size(adj2);
        LG = 31 - __builtin_clz(N);
        adj = adj2;
        tin = tout = vector<int>(N);
        up = st = vector (N, vector<int>(LG, neutral));
        up[root][0] = root;
        dfs(root);
    }

    bool ancestor(int u, int v) { return tin[u] <= tin[v] && tout[u] >=
       tout[v]; }

    int query2(int u, int v) {
        if (ancestor(u, v)) return neutral;
        int ans = neutral;
        for (int i = LG - 1; i >= 0; i--) {
            if (!ancestor(up[u][i], v)) {
                ans = merge(ans, st[u][i]);
                u = up[u][i];
```

```
            }
        }
        return merge(ans, st[u][0]);
    }

    int query(int u, int v) {
        if (u == v) return neutral;
#warning TRATAR ESSE CASO ACIMA
        return merge(query2(u, v), query2(v, u));
    }
} bl;
```

4.3 Binary Lifting Query (em nodos)

```
// Computa LCA e tambem resolve queries de operacoes
// associativas e comutativas em caminhos.
//
// Build(): O(n log(n))
// Query(): O(log(n))
// Lca(): O(log(n))
// Kth(): O(log(n))
//
// up[u][i] = (2 ^ i)-esimo pai do u
// st[u][i] = query ate (2 ^ i)-esimo pai do u (NAO INCLUI O U)

struct BinaryLifting {
    vector<vector<int>> adj, up, st;
    vector<int> val, tin, tout;
    int N, LG, t;

    const int neutral = 0;
    int merge(int l, int r) { return l + r; }

    void dfs(int u, int p = -1) {
        tin[u] = t++;
        for (int i = 0; i < LG - 1; i++) {
            up[u][i + 1] = up[up[u][i]][i];
            st[u][i + 1] = merge(st[u][i], st[up[u][i]][i]);
        }
        for (int v : adj[u]) if (v != p) {
            up[v][0] = u, st[v][0] = val[u];
            dfs(v, u);
        }
        tout[u] = t++;
    }

    void build(int root, vector<vector<int>>& adj2, vector<int>& v) {
        t = 1;
        N = size(adj2);
        LG = 31 - __builtin_clz(N);
        adj = adj2;
        val = v;
        tin = tout = vector<int>(N);
        up = st = vector (N, vector<int>(LG, neutral));
        up[root][0] = root;
        st[root][0] = val[root];
        dfs(root);
    }
```

```
    bool ancestor(int u, int v) { return tin[u] <= tin[v] && tout[u] >=
       tout[v]; }

    int query2(int u, int v, bool include_lca) {
        if (ancestor(u, v)) return include_lca ? val[u] : neutral;
        int ans = val[u];
        for (int i = LG - 1; i >= 0; i--) {
            if (!ancestor(up[u][i], v)) {
                ans = merge(ans, st[u][i]);
                u = up[u][i];
            }
        }
        return include_lca ? merge(ans, st[u][0]) : ans;
    }

    int query(int u, int v) {
        if (u == v) return val[u];
        return merge(query2(u, v, 1), query2(v, u, 0));
    }

    int lca(int u, int v) {
        if (ancestor(u, v)) return u;
        if (ancestor(v, u)) return v;
        for (int i = LG - 1; i >= 0; i--) {
            if (!ancestor(up[u][i], v)) u = up[u][i];
        }
        return up[u][0];
    }

    int kth(int u, int k) {
        for (int i = 0; i < LG; i++) {
            if (k & (1 << i)) u = up[u][i];
        }
        return u;
    }

} bl;
```

4.4 Binary Lifting Query (em nodos)

```
// Esse resolve queries de operacoes nao comutativas
// Levemente diferente do padrao
//
// Esse aqui resolve query de Kadani em arvore
// https://codeforces.com/contest/1843/problem/F2

struct node {
    int pref, suff, sum, best;
    node() : pref(0), suff(0), sum(0), best(0) {}
    node(int x) : pref(x), suff(x), sum(x), best(x) {}
    node(int a, int b, int c, int d) : pref(a), suff(b), sum(c), best(d) {}
};

node merge(node &l, node &r) {
    int pref = max(l.pref, l.sum + r.pref);
    int suff = max(r.suff, r.sum + l.suff);
    int sum = l.sum + r.sum;
    int best = max(l.suff + r.pref, max(l.best, r.best));
    return node(pref, suff, sum, best);
```

```
}

struct BinaryLifting {
    vector<vector<int>> adj, up;
    vector<int> val, tin, tout;
    vector<vector<node>> st, st2;
    int N, LG, t;

    void build(int u, int p = -1) {
        tin[u] = t++;
        for (int i = 0; i < LG - 1; i++) {
            up[u][i + 1] = up[up[u][i]][i];
            st[u][i + 1] = merge(st[u][i], st[up[u][i]][i]);
            st2[u][i + 1] = merge(st2[up[u][i]][i], st2[u][i]);
        }
        for (int v : adj[u])
            if (v != p) {
                up[v][0] = u;
                st[v][0] = node(val[u]);
                st2[v][0] = node(val[u]);
                build(v, u);
            }
        tout[u] = t++;
    }

    void build(int root, vector<vector<int>> &adj2, vector<int> &v) {
        t = 1;
        N = size(adj2);
        LG = 31 - __builtin_clz(N);
        adj = adj2;
        val = v;
        tin = tout = vector<int>(N);
        up = vector(N, vector<int>(LG));
        st = st2 = vector(N, vector<node>(LG));
        up[root][0] = root;
        st[root][0] = node(val[root]);
        st2[root][0] = node(val[root]);
        build(root);
    }

    bool ancestor(int u, int v) { return tin[u] <= tin[v] && tout[u] >=
       tout[v]; }

    node query2(int u, int v, bool include_lca, bool invert) {
        if (ancestor(u, v)) return include_lca ? node(val[u]) : node();
        node ans = node(val[u]);
        for (int i = LG - 1; i >= 0; i--) {
            if (!ancestor(up[u][i], v)) {
                if (invert)
                    ans = merge(st2[u][i], ans);
                else
                    ans = merge(ans, st[u][i]);
                u = up[u][i];
            }
        }
        if (!include_lca) return ans;
        return merge(ans, st[u][0]);
    }

    node query(int u, int v) {
        if (u == v) return node(val[u]);
```

```
        node l = query2(u, v, 1, 0);
        node r = query2(v, u, 0, 1);
        return merge(l, r);
    }

    int lca(int u, int v) {
        if (ancestor(u, v)) return u;
        if (ancestor(v, u)) return v;
        for (int i = LG - 1; i >= 0; i--) {
            if (!ancestor(up[u][i], v)) {
                u = up[u][i];
            }
        }
        return up[u][0];
    }

} bl, bl2;
```

4.5 Bridges e Edge Biconnected Components

```
// Acha todas as pontes em O(n)
// Tambem constroi a arvore condensada, mantendo
// so as pontes como arestas e o resto comprimindo
// em nodos
//
// Salva no vetor bridges os pares {u, v} cujas arestas sao pontes

typedef pair<int, int> ii;
const int maxn = 2e5 + 5;
int n, m;
bool vis[maxn];
int dp[maxn], dep[maxn];
vector<int> adj[maxn];
vector<ii> bridges;

void dfs_dp(int u, int p = -1, int d = 0) {
    dp[u] = 0, dep[u] = d, vis[u] = 1;
    for (auto v : adj[u]) {
        if (v != p) {
            if (vis[v]) {
                if (dep[v] < dep[u]) dp[v]--, dp[u]++;
            } else {
                dfs_dp(v, u, d + 1);
                dp[u] += dp[v];
            }
        }
    }
    if (dp[u] == 0 && p != -1) { // edge {u, p} eh uma ponte
        bridges.emplace_back(u, p);
    }
}

void find_bridges() {
    memset(vis, 0, n);
    for (int i = 0; i < n; i++) {
        if (!vis[i]) {
            dfs_dp(i);
        }
    }
```

```
}

// Edge Biconnected Components (requer todo codigo acima)

int ebcc[maxn], ncc = 0;
vector<int> adjbcc[maxn];

void dfs_ebcc(int u, int p, int cc) {
    vis[u] = 1;
    if (dp[u] == 0 && p != -1) {
        cc = ++ncc;
    }
    ebcc[u] = cc;
    for (auto v : adj[u]) {
        if (!vis[v]) {
            dfs_ebcc(v, u, cc);
        }
    }
}

void build_ebcc_graph() {
    find_bridges();
    memset(vis, 0, n);
    for (int i = 0; i < n; i++) {
        if (!vis[i]) {
            dfs_ebcc(i, -1, ncc);
            ++ncc;
        }
    }
    // Opcao 1 - constroi o grafo condensado passando por todas as edges
    for (int u = 0; u < n; u++) {
        for (auto v : adj[u]) {
            if (ebcc[u] != ebcc[v]) {
                adjbcc[ebcc[u]].emplace_back(ebcc[v]);
            } else {
                // faz algo
            }
        }
    }
    // Opcao 2 - constroi o grafo condensado passando so pelas pontes
    for (auto [u, v] : bridges) {
        adjbcc[ebcc[u]].emplace_back(ebcc[v]);
        adjbcc[ebcc[v]].emplace_back(ebcc[u]);
    }
}
```

4.6 Dinic

```
// Fonte: https://github.com/shahjalalshohag/code-library
//
// Max Flow em O(V^3) ou O(E * sqrt(V)) em bipartido

const int N = 5010;

const long long inf = 1LL << 61;
struct Dinic {
    struct edge {
        int to, rev;
        long long flow, w;
```

```
    int id;
};
int n, s, t, mxid;
vector<int> d, flow_through;
vector<int> done;
vector<vector<edge>> g;
Dinic() {}
Dinic(int _n) {
    n = _n + 10;
    mxid = 0;
    g.resize(n);
}
void add_edge(int u, int v, long long w, int id = -1) {
    edge a = {v, (int)g[v].size(), 0, w, id};
    edge b = {u, (int)g[u].size(), 0, 0, -2}; // for bidirectional edges
       cap(b) = w
    g[u].emplace_back(a);
    g[v].emplace_back(b);
    mxid = max(mxid, id);
}
bool bfs() {
    d.assign(n, -1);
    d[s] = 0;
    queue<int> q;
    q.push(s);
    while (!q.empty()) {
        int u = q.front();
        q.pop();
        for (auto &e : g[u]) {
            int v = e.to;
            if (d[v] == -1 && e.flow < e.w) d[v] = d[u] + 1, q.push(v);
        }
    }
    return d[t] != -1;
}
long long dfs(int u, long long flow) {
    if (u == t) return flow;
    for (int &i = done[u]; i < (int)g[u].size(); i++) {
        edge &e = g[u][i];
        if (e.w <= e.flow) continue;
        int v = e.to;
        if (d[v] == d[u] + 1) {
            long long nw = dfs(v, min(flow, e.w - e.flow));
            if (nw > 0) {
                e.flow += nw;
                g[v][e.rev].flow -= nw;
                return nw;
            }
        }
    }
    return 0;
}
long long max_flow(int _s, int _t) {
    s = _s;
    t = _t;
    long long flow = 0;
    while (bfs()) {
        done.assign(n, 0);
        while (long long nw = dfs(s, inf)) flow += nw;
    }
    flow_through.assign(mxid + 10, 0);
```

```
        for (int i = 0; i < n; i++)
            for (auto e : g[i])
                if (e.id >= 0) flow_through[e.id] = e.flow;
        return flow;
    }
};

/*
int main() {
    int n, m;
    cin >> n >> m;
    Dinic F(n + 1);
    for (int i = 1; i <= m; i++) {
        int u, v, w;
        cin >> u >> v >> w;
        F.add_edge(u, v, w);
    }
    cout << F.max_flow(1, n) << '\n';
    return 0;
}
*/
```

4.7 Pontos de articulacao

```
// Fonte: https://github.com/shahjalalshohag/code-library
//
// O equivalente a pontes, em vertices
//
// Complexidade: O(n)

const int N = 3e5 + 9;

int T, low[N], dis[N], art[N];
vector<int> g[N];
void dfs(int u, int pre = 0) {
    low[u] = dis[u] = ++T;
    int child = 0;
    for (auto v : g[u]) {
        if (!dis[v]) {
            dfs(v, u);
            low[u] = min(low[u], low[v]);
            if (low[v] >= dis[u] && pre != 0) art[u] = 1;
            ++child;
        } else if (v != pre)
            low[u] = min(low[u], dis[v]);
    }
    if (pre == 0 && child > 1) art[u] = 1;
}

/*
int32_t main() {
    ios_base::sync_with_stdio(0);
    cin.tie(0);
    while (1) {
        int n, m;
        cin >> n >> m;
        if (!n) break;
        while (m--) {
            int u, v;
```

```
            cin >> u >> v;
            g[u].push_back(v);
            g[v].push_back(u);
        }
        dfs(1);
        int ans = 0;
        for (int i = 1; i <= n; i++) ans += art[i];
        cout << ans << '\n';
        T = 0;
        for (int i = 1; i <= n; i++) low[i] = dis[i] = art[i] = 0,
           g[i].clear();
    }
    return 0;
}
*/
```

5 Matematica

5.1 Crivo de Eratostenes

```
// Computa numeros primos entre [2, n] em O(n)
//
// Crivo linear computando spf (smallest prime factor) pra cada numero
// x entre [2, n] e phi(x) (funcao totiente)
// Complexidade: O(n)

int spf[maxn], phi[maxn];
vector<int> primes;
void sieve(int n) {
    phi[1] = 1;
    for (int i = 2; i <= n; i++) {
        if (spf[i] == 0) {
            spf[i] = i;
            primes.emplace_back(i);
            phi[i] = i - 1;
        }
        for (int j = 0; j < (int)primes.size() && i * primes[j] <= n &&
           primes[j] <= spf[i]; j++) {
            spf[i * primes[j]] = primes[j];
            if (primes[j] < spf[i])
                phi[i * primes[j]] = phi[i] * phi[primes[j]];
            else
                phi[i * primes[j]] = phi[i] * primes[j];
        }
    }
}
```

5.2 Fast Fourier Transform

```
// Fonte: https://github.com/ShahjalalShohag/code-library
//
// Faz convolucao de dois polinomios
// Complexidade: O(n log(n))
//
// Testado e sem erro de precisao para MAXN = 3e5 e A_i = 1e9

const int N = 3e5 + 9;

const double PI = acos(-1);
struct base {
    double a, b;
    base(double a = 0, double b = 0) : a(a), b(b) {}
    const base operator+(const base &c) const { return base(a + c.a, b +
       c.b); }
    const base operator-(const base &c) const { return base(a - c.a, b -
       c.b); }
    const base operator*(const base &c) const { return base(a * c.a - b *
       c.b, a * c.b + b * c.a); }
};
void fft(vector<base> &p, bool inv = 0) {
    int n = p.size(), i = 0;
    for (int j = 1; j < n - 1; ++j) {
        for (int k = n >> 1; k > (i ^= k); k >>= 1)
            ;
        if (j < i) swap(p[i], p[j]);
```

```
    }
    for (int l = 1, m; (m = l << 1) <= n; l <<= 1) {
        double ang = 2 * PI / m;
        base wn = base(cos(ang), (inv ? 1. : -1.) * sin(ang)), w;
        for (int i = 0, j, k; i < n; i += m) {
            for (w = base(1, 0), j = i, k = i + l; j < k; ++j, w = w * wn) {
                base t = w * p[j + l];
                p[j + l] = p[j] - t;
                p[j] = p[j] + t;
            }
        }
    }
    if (inv)
        for (int i = 0; i < n; ++i) p[i].a /= n, p[i].b /= n;
}
vector<long long> multiply(vector<int> &a, vector<int> &b) {
    int n = a.size(), m = b.size(), t = n + m - 1, sz = 1;
    while (sz < t) sz <<= 1;
    vector<base> x(sz), y(sz), z(sz);
    for (int i = 0; i < sz; ++i) {
        x[i] = i < (int)a.size() ? base(a[i], 0) : base(0, 0);
        y[i] = i < (int)b.size() ? base(b[i], 0) : base(0, 0);
    }
    fft(x), fft(y);
    for (int i = 0; i < sz; ++i) z[i] = x[i] * y[i];
    fft(z, 1);
    vector<long long> ret(sz);
    for (int i = 0; i < sz; ++i) ret[i] = (long long)round(z[i].a);
    while ((int)ret.size() > 1 && ret.back() == 0) ret.pop_back();
    return ret;
}

/*
long long ans[N];
int32_t main() {
    ios_base::sync_with_stdio(0);
    cin.tie(0);
    int n, x;
    cin >> n >> x;
    vector<int> a(n + 1, 0), b(n + 1, 0), c(n + 1, 0);
    int nw = 0;
    a[0]++;
    b[n]++;
    long long z = 0;
    for (int i = 1; i <= n; i++) {
        int k;
        cin >> k;
        nw += k < x;
        a[nw]++;
        b[-nw + n]++;
        z += c[nw] + !nw;
        c[nw]++;
    }
    auto res = multiply(a, b);
    for (int i = n + 1; i < res.size(); i++) {
        ans[i - n] += res[i];
    }
    ans[0] = z;
    for (int i = 0; i <= n; i++) cout << ans[i] << ' ';
    cout << '\n';
    return 0;
```

```
}
*/
```

5.3 Pollard Rho

```
// Fonte: https://github.com/shahjalalshohag/code-library
//
// Fatora numeros ate 8*10^18
// Complexidade: O(n ^ (1/4))

namespace PollardRho {
    mt19937 rnd(chrono::steady_clock::now().time_since_epoch().count());
    const int P = 1e6 + 9;
    ll seq[P];
    int primes[P], spf[P];
    inline ll add_mod(ll x, ll y, ll m) { return (x += y) < m ? x : x - m; }
    inline ll mul_mod(ll x, ll y, ll m) {
        ll res = __int128(x) * y % m;
        return res;
        // ll res = x * y - (ll)((long double)x * y / m + 0.5) * m;
        // return res < 0 ? res + m : res;
    }
    inline ll pow_mod(ll x, ll n, ll m) {
        ll res = 1 % m;
        for (; n; n >>= 1) {
            if (n & 1) res = mul_mod(res, x, m);
            x = mul_mod(x, x, m);
        }
        return res;
    }
    // O(it * (logn)^3), it = number of rounds performed
    inline bool miller_rabin(ll n) {
        if (n <= 2 || (n & 1 ^ 1)) return (n == 2);
        if (n < P) return spf[n] == n;
        ll c, d, s = 0, r = n - 1;
        for (; !(r & 1); r >>= 1, s++) {
        }
        // each iteration is a round
        for (int i = 0; primes[i] < n && primes[i] < 32; i++) {
            c = pow_mod(primes[i], r, n);
            for (int j = 0; j < s; j++) {
                d = mul_mod(c, c, n);
                if (d == 1 && c != 1 && c != (n - 1)) return false;
                c = d;
            }
            if (c != 1) return false;
        }
        return true;
    }
    void init() {
        int cnt = 0;
        for (int i = 2; i < P; i++) {
            if (!spf[i]) primes[cnt++] = spf[i] = i;
            for (int j = 0, k; (k = i * primes[j]) < P; j++) {
                spf[k] = primes[j];
                if (spf[i] == spf[k]) break;
            }
        }
    }
```

```
    // returns O(n^(1/4))
    ll pollard_rho(ll n) {
        while (1) {
            ll x = rnd() % n, y = x, c = rnd() % n, u = 1, v, t = 0;
            ll *px = seq, *py = seq;
            while (1) {
                *py++ = y = add_mod(mul_mod(y, y, n), c, n);
                *py++ = y = add_mod(mul_mod(y, y, n), c, n);
                if ((x = *px++) == y) break;
                v = u;
                u = mul_mod(u, abs(y - x), n);
                if (!u) return __gcd(v, n);
                if (++t == 32) {
                    t = 0;
                    if ((u = __gcd(u, n)) > 1 && u < n) return u;
                }
            }
            if (t && (u = __gcd(u, n)) > 1 && u < n) return u;
        }
    }

    vector<ll> factorize(ll n) {
        if (n == 1) return vector<ll>();
        if (miller_rabin(n)) return vector<ll>{n};
        vector<ll> v, w;
        while (n > 1 && n < P) {
            v.push_back(spf[n]);
            n /= spf[n];
        }
        if (n >= P) {
            ll x = pollard_rho(n);
            v = factorize(x);
            w = factorize(n / x);
            v.insert(v.end(), w.begin(), w.end());
        }
        return v;
    }
} // namespace PollardRho

/*
int32_t main() {
    ios_base::sync_with_stdio(0);
    cin.tie(0);
    PollardRho::init();
    int t;
    cin >> t;
    while (t--) {
        ll n;
        cin >> n;
        auto f = PollardRho::factorize(n);
        sort(f.begin(), f.end());
        cout << f.size() << ' ';
        for (auto x : f) cout << x << ' ';
        cout << '\n';
    }
    return 0;
}
*/
```

6 Geometria

6.1 Geometria inteiro

```
// Tudo que temos de geometria pra pontos inteiros

// Ponto com coordenadas inteiras e alguns metodos

 struct pt {
     ll x, y;
     pt() : x(0), y(0) {}
     pt(ll _x, ll _y) : x(_x), y(_y) {}

     pt operator*(const ll &b) { return pt(b * x, b * y); }
     pt operator-(const pt &b) { return pt(x - b.x, y - b.y); }
     pt operator+(const pt &b) { return pt(x + b.x, y + b.y); }
     ll operator*(const pt &b) { return x * b.x + y * b.y; }
     ll operator^(const pt &b) { return x * b.y - y * b.x; }

     bool operator<(const pt &p) const {
         if (x == p.x) return y < p.y;
         return x < p.x;
     }
     ll dist2(const pt &p) {
         ll dx = x - p.x;
         ll dy = y - p.y;
         return dx * dx + dy * dy;
     }

     friend ostream &operator<<(ostream &out, const pt &a) { return out << "("
        << a.x << "," << a.y << ")"; }
     friend istream &operator>>(istream &in, pt &a) { return in >> a.x >> a.y;
         }
 };

 // Convex Hull
 // Algoritmo Graham's Scan
 // Complexidade: O(n log(n))

 bool ccw(pt &p, pt &a, pt &b, bool collinear = 0) {
     pt p1 = a - p;
     pt p2 = b - p;
     return collinear ? (p2 ^ p1) <= 0 : (p2 ^ p1) < 0;
 }

 void sort_by_angle(vector<pt>& v) { // sorta o vetor por angulo em relacao ao
    pivo
     pt p0 = *min_element(begin(v), end(v));
     sort(begin(v), end(v), [&](pt &l, pt &r) { // sorta clockwise
         pt p1 = l - p0;
         pt p2 = r - p0;
         ll c1 = p1 ^ p2;
         return c1 < 0 || ((c1 == 0) && p0.dist2(l) < p0.dist2(r));
     });
 }

 vector<pt> convex_hull(vector<pt> v, bool collinear = 0) {
     int n = size(v);

     sort_by_angle(v);
```

```
    if (collinear) {
        for (int i = n - 2; i >= 0; i--) { // reverte o ultimo lado do
            poligono
            if (ccw(v[0], v[n - 1], v[i])) {
                reverse(begin(v) + i + 1, end(v));
                break;
            }
        }
    }

    vector<pt> ch{v[0], v[1]};
    for (int i = 2; i < n; i++) {
        while (ch.size() > 2 && (ccw(ch.end()[-2], ch.end()[-1], v[i],
            !collinear))) ch.pop_back();
        ch.emplace_back(v[i]);
    }

    return ch;
}
```

7 Extra

7.1 Config do Vim

```
// .vimrc

 set nu
 set ai
 set ts=4
 set sw=4
 set so=10
 filetype plugin indent on
 inoremap {} {}<Left><Return><Up><End><Return>

 au BufReadPost * if line("'\"") > 0 && line("'\"") <= line("$") | exe
    "normal! g'\"" | endif

 set nohls
 set belloff=all
 syntax on
 set expandtab
 set noshiftround
 set showmode
 set showcmd
```

7.2 Custom Hash

```
// Hash personalizado pra evitar colisao no unordered_map
// Uso: map<int, int, custom_hash> mapa;

 struct custom_hash {
     static uint64_t splitmix64(uint64_t x) {
         x += 0x9e3779b97f4a7c15;
         x = (x ^ (x >> 30)) * 0xbf58476d1ce4e5b9;
         x = (x ^ (x >> 27)) * 0x94d049bb133111eb;
         return x ^ (x >> 31);
     }

     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);
     }
 };
```

7.3 Gerador aleatorio de casos

```
 #include <bits/stdc++.h>
 using namespace std;
 typedef long long ll;
 mt19937 rng(chrono::steady_clock::now().time_since_epoch().count());

 ll uniform(ll l, ll r) {
     uniform_int_distribution<int> uid(l, r);
     return uid(rng);
 }
```

```
int main(){
    cout << uniform(1, 10) << endl;
}
```

7.4 Mint easier

```
// Inteiro automaticamente modulado
// Bem compacto

struct mint {
    int val;
    int fix(ll x) { return ((x % mod) + mod) % mod; }
    mint(ll v = 0) { val = fix(v); }
    mint pwr(mint b, ll e) {
        mint res;
        for (res = 1; e; e >>= 1, b = b * b) if (e & 1) res = res * b;
        return res;
    }
    bool operator==(mint o) { return val == o.val; }
    friend mint operator*(mint a, mint o) { return a.fix((ll)a.val * o.val); }
    friend mint operator+(mint a, mint o) { return a.fix(a.val + o.val); }
    friend mint operator-(mint a, mint o) { return a.fix(a.val - o.val); }
    friend mint operator^(mint a, ll o) { return a.pwr(a, o); }
    friend mint operator/(mint a, mint o) { return a * (o ^ (mod - 2)); }
};
```

7.5 Mint faster

```
// Inteiro automaticamente modulado
// Tem um pouco mais de codigo, mas eh mais rapido

const int mod = 998244353;

struct mint {
    int val;
    mint(ll v = 0) { val = v % mod; if (val < 0) val += mod; }
    mint pwr(mint b, ll e) {
        mint res;
        for (res = 1; e; e >>= 1, b = b * b) if (e & 1) res = res * b;
        return res;
    }
    bool operator==(mint o) { return val == o.val; }
    friend mint operator*(mint a, mint o) { return (ll)a.val * o.val % mod; }
    friend mint operator+(mint a, mint o) {
        a.val += o.val;
        if (a.val >= mod) a.val -= mod;
        return a;
    }
    friend mint operator-(mint a, mint o) {
        a.val -= o.val;
        if (a.val < 0) a.val += mod;
        return a;
    }
    friend mint operator^(mint a, ll o) { return a.pwr(a, o); }
    friend mint operator/(mint a, mint o) { return a * (o ^ (mod - 2)); }
};
```

7.6 Rand C++

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

7.7 Script de stress test

```
set -e
g++-13 -O2 code.cpp -o code
g++-13 -O2 brute.cpp -o brute
g++-13 -O2 gen.cpp -o gen

for((i = 1; ; ++i)); do
    ./gen > in
    ./code < in > myout
    ./brute < in > out
    diff myout out > /dev/null || break
    echo "OK: "  $i
done

echo "WA:"
cat in
echo "Myout:"
cat myout
echo "Out:"
cat out
```

7.8 Script pra rodar C++

```
// chmod +x run
// ./run A.cpp

#!/bin/bash
g++ --std=c++20 -Wall -O2 -DNTJ -fsanitize=address,undefined $1 && ./a.out
```

7.9 Template C++

```
#include <bits/stdc++.h>
#define endl '\n'

using namespace std;
typedef long long ll;

void solve() {}

signed main() {
    ios_base::sync_with_stdio(0);
    cin.tie(0);
    solve();
}
```

7.10 Template de debug simples

```
void _print() {}
template <typename T, typename... U> void _print(T a, U... b) {
    if (sizeof...(b)) {
        cerr << a << ", ";
        _print(b...);
    } else
        cerr << a;
}
#ifdef NTJ
#define debug(x...) cerr << "[" << #x << "] = [", _print(x), cerr << "]" <<
    endl
#else
#define debug(...)
#endif
```