[UNICAMP] Moratonistas

Bernardo Archegas, Fernando Morato e Luiz Oda

Índice

1 Estruturas

1.1 Compressao de Coordenadas

```
// Classe para comprimir coordenadas

// d8283d

template <typename T>
class CoordinateCompression {
public:
    CoordinateCompression(const std::vector<T> &_v) {
        v = _v;
        std::sort(v.begin(), v.end());
        v.resize(std::unique(v.begin(), v.end()) -
            v.begin());
    }

    int size() { return (int) v.size(); }

    int operator() (T x) {
        return (int) (std::lower_bound(v.begin(), v.end(),
            x) - v.begin());
    }
private:
    std::vector<T> v;
};
```

1.2 Fenwick Tree

```
// 29985a
template <typename T>
class FenwickTree {
public:
    void init(int _n) {
        n = _n;
        bit.assign(n + 1, 0);
    }

    void init(vector<T> &v) {
        n = (int) v.size();
        bit.assign(n + 1, 0);
        for (int i = 1; i <= n; i++) {
            bit[i] += v[i - 1];
            if (i + (i & -i) <= n)
                bit[i + (i & -i)] += bit[i];
        }
    }

    void update(int i, T x) {
        for (; i <= n; i += i & -i)
            bit[i] += x;
    }

    T query(int i) {
        T ans = 0;
        for (; i > 0; i -= i & -i)
            ans += bit[i];
        return ans;
    }

private:
    int n;
```

```
    std::vector<T> bit;
};
```

1.3 Otimizacao para Mo - Hilbert

```
// a1f430
namespace MO {
    constexpr int logn = 20;
    constexpr int maxn = 1 << logn;

    long long hilbertorder(int x, int y) {
        long long d = 0;
        for (int s = 1 << (logn - 1); s; s >>= 1) {
            bool rx = x & s, ry = y & s;
            d = ((((d << 2) | rx) * 3) ^
                static_cast<int>(ry));
            if (!ry) {
                if (rx) {
                    x = maxn - x;
                    y = maxn - y;
                }
                swap(x, y);
            }
        }
        return d;
    }

    template <class T>
    void sortQueries(vector<T> &v) {
        for (auto &x : v)
            x.id = hilbertorder(x.l, x.r);
        sort(v.begin(), v.end());
    }

    struct Query {
        // maybe add new stuff here
        long long id;
        int l, r, idx;

        // Query(int _idx, int _l, int _r): idx(_idx),
            l(_l), r(_r) {}

        bool operator < (const Query &o) const {
            return id < o.id;
        }
    };
}

using namespace MO;
```

1.4 Policy Based Data Structure

```
// d501f8
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>

template<typename T, typename M = __gnu_pbds::null_type>
using ordered_set = __gnu_pbds::tree<T, M, less<T>,
    __gnu_pbds::rb_tree_tag,
    __gnu_pbds::tree_order_statistics_node_update>;

// Supports the same operations as the std::set

// find_by_order(k) - Returns an iterator to the k-th
    largest element (counting from 0)

// order_of_key(x) - Returns the number of items in a set
    that are strictly smaller than x
```

1.5 Segment Tree

```
// 01129d
struct Node {
    // atributes

    Node() {
        // empty node
    }
```

```
    Node() {
        // constructor
    }

    Node(const Node &l, const Node &r) {
        // merge
    }
};

template <class node_t, class e_t>
class SegTree {
public:
    void init(vector<e_t> &v) {
        n = (int) v.size();
        tree.resize(2 * n);
        for (int i = 0; i < n; i++) {
            tree[i + n] = node_t(v[i]);
        }
        for (int i = n - 1; i > 0; i--) {
            tree[i] = node_t(tree[2 * i], tree[2 * i + 1]);
        }
    }

    void update(int p, e_t x) {
        p += n;
        tree[p] = node_t(x);
        for (p /= 2; p > 0; p /= 2) {
            tree[p] = node_t(tree[2 * p], tree[2 * p + 1]);
        }
    }

    node_t query(int l, int r) {
        node_t ln, rn;
        for (l += n, r += n; l < r; l /= 2, r /= 2) {
            if (l & 1) ln = node_t(ln, tree[l++]);
            if (r & 1) rn = node_t(tree[--r], rn);
        }
        return node_t(ln, rn);
    }

private:
    int n;
    std::vector<node_t> tree;
};
```

1.6 Segment Tree com Lazy

```
// 51aeac
struct Lazy {
    // atributes

    Lazy() {
        // constructor
    }

    void reset() {

    }

    void operator += (Lazy o) {
        // merge
    }
};

struct Node {
    // atributes

    Node() {
        // empty node
    }

    Node() {
        // constructor
    }

    Node(const Node &l, const Node &r) {
        // merge
    }

    void apply(Lazy lazy) {
```

```
    }
};

// may be changed to iterative
template <class node_t, class e_t, class lazy_t = int>
class SegTree {
public:
    void init(vector<e_t> &_v) {
        v = _v;
        n = (int) v.size();
        tree.resize(4 * n + 1);
        lazy.resize(4 * n + 1);
        dirty.assign(4 * n + 1, false);
        build(1, 0, n - 1);
    }

    void update(int node, int l, int r, int ql, int qr,
       lazy_t x) {
        push(node, l, r);
        if (ql > r || l > qr)
            return;
        if (ql <= l && r <= qr) {
            apply(node, l, r, x);
            push(node, l, r);
            return;
        }
        int m = (l + r) / 2;
        update(2 * node, l, m, ql, qr, x);
        update(2 * node + 1, m + 1, r, ql, qr, x);
        tree[node] = node_t(tree[2 * node], tree[2 * node +
           1]);
    }

    node_t query(int node, int l, int r, int ql, int qr) {
        push(node, l, r);
        if (ql > r || l > qr)
            return node_t();
        if (ql <= l && r <= qr)
            return tree[node];
        int m = (l + r) / 2;
        return node_t(query(2 * node, l, m, ql, qr), query(2
           * node + 1, m + 1, r, ql, qr));
    }
private:
    int n;
    std::vector<e_t> v;
    std::vector<bool> dirty;
    std::vector<node_t> tree;
    std::vector<lazy_t> lazy;

    void apply(int node, int l, int r, Lazy &lz) {
        tree[node].apply(lz);
        if (l != r) {
            dirty[node] = true;
            lazy[node] += lz;
        }
    }

    void push(int node, int l, int r) {
        if (dirty[node]) {
            int m = (l + r) / 2;
            apply(2 * node, l, m, lazy[node]);
            apply(2 * node + 1, m + 1, r, lazy[node]);
            lazy[node].reset();
            dirty[node] = false;
        }
    }

    void build(int node, int l, int r) {
        if (l == r)  {
            tree[node] = node_t(v[l]);
        } else {
            int m = (l + r) / 2;
            build(2 * node, l, m);
            build(2 * node + 1, m + 1, r);
            tree[node] = node_t(tree[2 * node], tree[2 *
               node + 1]);
        }
    }
};
```

1.7 Segment Tree Persistente

```cpp
// f9bdcf
struct Node{
    int v = 0;
    Node *l = this, *r = this;
};

int CNT = 1;
Node buffer[MAXN * 20];

Node* update(Node *root, int l, int r, int idx, int val){
    Node *node = &buffer[CNT++];
    *node = *root;
    int mid = (l + r) / 2;
    node->v += val;
    if(l + 1 != r){
        if(idx < mid) node->l = update(root->l, l, mid, idx,
            val);
        else node->r = update(root->r, mid, r, idx, val);
    }
    return node;
}

int query(Node *node, int tl, int tr, int l, int r){
    if(l <= tl && tr <= r) return node->v;
    if(tr <= l || tl >= r) return 0;
    int mid = (tl + tr) / 2;
    return query(node->l, tl, mid, l, r) + query(node->r,
        mid, tr, l, r);
}
```

1.8 Sparse Table

```cpp
// Based on Juvitus and tfg implementations
// 21b1d0
template <class T, class F = function<T(const T&, const T&)>>
class SparseTable {
public:
    void init(const vector<T> &v, const F &_f) {
        f = _f;
        n = (int) v.size();
        int l = 0;
        while ((1 << l) / 2 < n) {
            l++;
        }
        pw.assign(n + 1, -1);
        tab.assign(l, vector<T>(n));
        for (int i = 0; i < n; i++) {
            tab[0][i] = v[i];
            pw[i + 1] = pw[(i + 1) / 2] + 1;
        }
        for (int i = 0; i + 1 < l; i++) {
            for (int j = 0; j + (1 << i) < n; j++) {
                tab[i + 1][j] = f(tab[i][j], tab[i][j + (1
                    << i)]);
            }
        }
    }

    T query(int l, int r) {
        int k = pw[r - l];
        return f(tab[k][l], tab[k][r - (1 << k)]);
    }

private:
    F f;
    int n;
    vector<int> pw;
    vector<vector<T>> tab;
};
```

1.9 Treap Implicita

```cpp
// 2f6de4
namespace treap {
    // mt19937 rng((unsigned int)
        chrono::steady_clock::now().time_since_epoch().count());

    struct node {
        node *l, *r;
        int value;
```

```
    int prior, sz;
    bool rev;

    node (int _v): value(_v) {
        sz = 1;
        prior = rng();
        rev = false;
        l = r = NULL;
    }
};

inline int getSize(node *nd) {
    return (nd ? nd->sz : 0);
}

inline void update(node *&nd) {
    nd->sz = getSize(nd->l) + getSize(nd->r) + 1;
}

void push(node *nd) {
    if (nd && nd->rev) {
        nd->rev = false;
        swap(nd->l, nd->r);
        if (nd->l) {
            nd->l->rev ^= true;
        }
        if (nd->r) {
            nd->r->rev ^= true;
        }
    }
}

void merge(node *&root, node *l, node *r) {
    push(l), push(r);
    if (!l || !r) {
        root = (l ? l : r);
    } else {
        if (l->prior > r->prior) {
            merge(l->r, l->r, r);
            root = l;
        } else {
            merge(r->l, l, r->l);
            root = r;
        }
        update(root);
    }
}

void split(node *root, node *&l, node *&r, int pos) {
    if (!root) {
        l = r = NULL;
    } else {
        push(root);
        int p = getSize(root->l) + 1;
        if (p <= pos) {
            split(root->r, root->r, r, pos - p);
            l = root;
        } else {
            split(root->l, l, root->l, pos);
            r = root;
        }
        update(root);
    }
}

void insert(node *&root, int value, int pos) {
    node *l = NULL, *r = NULL, *aux = new node(value);
    split(root, l, r, pos);
    merge(l, l, aux);
    merge(root, l, r);
}

void reverse(node *&root, int ll, int rr) {
    // reverses interval [ll, rr]
    node *l = NULL, *r = NULL, *k = NULL;
    split(root, l, r, ll - 1);
    split(r, k, r, rr - ll + 1);
    k->rev ^= true;
    merge(r, k, r);
    merge(root, l, r);
}

void shift(node *&root, int ll, int rr) {
    // right-cyclic-shift on interval [ll, rr]
```

```
        node *l = NULL, *r = NULL, *k = NULL, *t = NULL;
        split(root, l, r, ll - 1);
        split(r, k, r, rr - ll + 1);
        split(k, k, t, rr - ll);
        merge(k, t, k);
        merge(r, k, r);
        merge(root, l, r);
    }

    int query(node *&root, int pos) {
        // returns which node is at position "pos"
        node *l = NULL, *r = NULL, *k = NULL;
        split(root, l, r, pos);
        split(l, l, k, pos - 1);
        int ans = k->value;
        merge(l, l, k);
        merge(root, l, r);
        return ans;
    }
}
```

1.10 Union Find

```
// 739726
class DSU {
public:
    void init(int n) {
        p.resize(n);
        std::iota(p.begin(), p.end(), 0);
        sz.assign(n, 1);
    }

    int getSize(int x) { return sz[x]; }

    int find(int x) { return x == p[x] ? x : p[x] =
       find(p[x]); }

    bool sameSet(int a, int b) { return find(a) == find(b); }

    void join(int a, int b) {
        a = find(a), b = find(b);
        if (a != b) {
            if (sz[a] > sz[b])
                std::swap(a, b);
            op.emplace_back(a, p[a]);
            os.emplace_back(b, sz[b]);
            p[a] = b;
            sz[b] += sz[a];
        }
    }

    void rollback() {
        assert(!op.empty() && !os.empty());
        p[op.back().first] = op.back().second;
        op.pop_back();
        sz[os.back().first] = os.back().second;
        os.pop_back();
    }

private:
    std::vector<int> p, sz;
    std::vector<std::pair<int, int>> op, os;
};
```

2 Geometria

2.1 Convex Hull

```
// 4c51e2
vector<PT> convexHull(vector<PT> &p, bool sorted = false) {
    int n = (int) p.size(), k = 0;
    if (n == 1)
        return p;
    if (!sorted)
        sort(p.begin(), p.end());
    vector<PT> h(2 * n + 1);
    // Upper-Hull
    for (int i = 0; i < n; i++) {
```

```
        while (k >= 2 && (p[i] - h[k - 2]) % (h[k - 1] - h[k
            - 2]) >= 0)
            k--;
        h[k++] = p[i];
    }
    // Lower-Hull
    for (int i = n - 2, t = k + 1; i >= 0; i--) {
        while (k >= t && (p[i] - h[k - 2]) % (h[k - 1] - h[k
            - 2]) >= 0)
            k--;
        h[k++] = p[i];
    }
    h.resize(k - 1);
    return h;
}
```

2.2 Ploblema dos pares mais proximos

```
// a673cd
pii closestPair(vector<PT> &p) {
    auto sqDist = [](PT pt) { return pt * pt; };
    long long dist = sqDist(p[0] - p[1]);
    pii ans(0, 1);
    int n = (int) p.size();
    vector<int> v(n);
    iota(v.begin(), v.end(), 0);
    sort(v.begin(), v.end(), [&](int a, int b) { return
        p[a].x < p[b].x; });
    set<pll> st;
    auto sq = [](long long x) { return x * x; };
    for (int l = 0, r = 0; r < n; r++) {
        while (sq(p[v[l]].x - p[v[r]].x) > dist) {
            st.erase(pll(p[v[l]].y, v[l]));
            l++;
        }
        long long delta = (ll) sqrt(dist) + 1;
        auto il = st.lower_bound(pll(p[v[r]].y - delta, -1));
        auto ir = st.lower_bound(pll(p[v[r]].y + delta, n +
            1));
        for (auto it = il; it != ir; it++) {
            long long distNow = sqDist(p[v[r]] -
                p[it->second]);
            if (distNow < dist) {
                dist = distNow;
                ans = pii(v[r], it->second);
            }
        }
        st.insert(pll(p[v[r]].y, v[r]));
    }
    if (ans.first > ans.second)
        swap(ans.first, ans.second);
    return ans;
}
```

2.3 Ponto 2D

```
// 198381

// template <typename T>
struct PT {
    #define T long long
    T x, y;
    PT(T _x = 0, T _y = 0): x(_x), y(_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 * (T c)             const { return PT(c * x,
        c * y); }
    T operator * (const PT &p)      const { return x * p.x +
        y * p.y; }
    T operator % (const PT &p)      const { return x * p.y -
        y * p.x; }
    bool operator < (const PT &p)   const { return x == p.x
        ? y < p.y : x < p.x; }
    bool operator == (const PT &p)  const { return x == p.x
        && y == p.y; }

    friend std::ostream& operator << (std::ostream &os,
        const PT &p) {
```

```
        return os << p.x << ' ' << p.y;
    }
    friend std::istream& operator >> (std::istream &is, PT
        &p) {
        return is >> p.x >> p.y;
    }
};
```

3 Grafos

3.1 Algoritmo Hungaro

```
// Resolve o problema de assignment (matriz n x n)
// Colocar os valores da matriz em 'a' (pode < 0)
// assignment() retorna um par com o valor do
// assignment minimo, e a coluna escolhida por cada linha
//
// O(n^3)
// 64c53e

// copied from:
   https://github.com/brunomaletta/Biblioteca/blob/master/Codigo/Problemas/hungarian.cpp

template<typename T> struct hungarian {
    int n;
    vector<vector<T>> a;
    vector<T> u, v;
    vector<int> p, way;
    T inf;

    hungarian(int n_) : n(n_), u(n+1), v(n+1), p(n+1),
        way(n+1) {
        a = vector<vector<T>>(n, vector<T>(n));
        inf = numeric_limits<T>::max();
    }
    pair<T, vector<int>> assignment() {
        for (int i = 1; i <= n; i++) {
            p[0] = i;
            int j0 = 0;
            vector<T> minv(n+1, inf);
            vector<int> used(n+1, 0);
            do {
                used[j0] = true;
                int i0 = p[j0], j1 = -1;
                T delta = inf;
                for (int j = 1; j <= n; j++) if (!used[j]) {
                    T cur = a[i0-1][j-1] - u[i0] - v[j];
                    if (cur < minv[j]) minv[j] = cur, way[j]
                        = j0;
                    if (minv[j] < delta) delta = minv[j], j1
                        = j;
                }
                for (int j = 0; j <= n; j++)
                    if (used[j]) u[p[j]] += delta, v[j] -=
                        delta;
                    else minv[j] -= delta;
                j0 = j1;
            } while (p[j0] != 0);
            do {
                int j1 = way[j0];
                p[j0] = p[j1];
                j0 = j1;
            } while (j0);
        }
        vector<int> ans(n);
        for (int j = 1; j <= n; j++) ans[p[j]-1] = j-1;
        return make_pair(-v[0], ans);
    }
};
```

3.2 Binary Lifting

```
// 1e18d0
namespace BinaryLifting {
    const int MAXN = 2e5 + 5;
    const int LOG = 20;

    int h[MAXN];
    int anc[LOG][MAXN];
```

```
void dfs(vector<vector<int>> &adj, int on, int par, int
    lvl = 0) {
    h[on] = lvl;
    anc[0][on] = par;
    for (int to : adj[on]) {
        if (to != par) {
            dfs(adj, to, on, lvl + 1);
        }
    }
}

void init(vector<vector<int>> &adj, int start = 0) {
    dfs(adj, start, start);
    for (int i = 1; i < LOG; i++) {
        for (int j = 0; j < (int) adj.size(); j++) {
            anc[i][j] = anc[i - 1][anc[i - 1][j]];
        }
    }
}

void up(int &x, int d) {
    for (int i = 0; i < LOG; i++) {
        if (d & (1 << i))
            x = anc[i][x];
    }
}

int getLCA(int a, int b) {
    if (h[a] > h[b]) {
        swap(a, b);
    }
    up(b, h[b] - h[a]);
    if (a == b) {
        return a;
    }
    for (int i = LOG - 1; i >= 0; i--) {
        if (anc[i][a] != anc[i][b]) {
            a = anc[i][a];
            b = anc[i][b];
        }
    }
    return anc[0][a];
}

int dist(int a, int b) {
    return h[a] + h[b] - 2 * h[getLCA(a, b)];
}
}
```

3.3 Dinic - Max Flow

```
// b39b81

// by tfg50
// found at:
   https://github.com/tfg50/Competitive-Programming/blob/master/Bi
template <class T = int>
class Dinic {
public:
    struct Edge {
        Edge(int a, T b){to = a;cap = b;}
        int to;
        T cap;
    };

    Dinic(int _n) : n(_n) {
        edges.resize(n);
    }

    T maxFlow(int src, int sink) {
        T ans = 0;
        while(bfs(src, sink)) {
            // maybe random shuffle edges against bad cases?
            T flow;
            pt = std::vector<int>(n, 0);
            while((flow = dfs(src, sink))) {
                ans += flow;
            }
        }
        return ans;
    }
```

```
    void addEdge(int from, int to, T cap, T other = 0) {
        edges[from].push_back(list.size());
        list.push_back(Edge(to, cap));
        edges[to].push_back(list.size());
        list.push_back(Edge(from, other));
    }

    bool inCut(int u) const { return h[u] < n; }
    int size() const { return n; }
private:
    int n;
    std::vector<std::vector<int> > edges;
    std::vector<Edge> list;
    std::vector<int> h, pt;

    T dfs(int on, int sink, T flow = 1e9) {
        if(flow == 0) {
            return 0;
        } if(on == sink) {
            return flow;
        }
        for(; pt[on] < (int) edges[on].size(); pt[on]++) {
            int cur = edges[on][pt[on]];
            if(h[on] + 1 != h[list[cur].to]) {
                continue;
            }
            T got = dfs(list[cur].to, sink, std::min(flow,
               list[cur].cap));
            if(got) {
                list[cur].cap -= got;
                list[cur ^ 1].cap += got;
                return got;
            }
        }
        return 0;
    }

    bool bfs(int src, int sink) {
        h = std::vector<int>(n, n);
        h[src] = 0;
        std::queue<int> q;
        q.push(src);
        while(!q.empty()) {
            int on = q.front();
            q.pop();
            for(auto a : edges[on]) {
                if(list[a].cap == 0) {
                    continue;
                }
                int to = list[a].to;
                if(h[to] > h[on] + 1) {
                    h[to] = h[on] + 1;
                    q.push(to);
                }
            }
        }
        return h[sink] < n;
    }
};
```

3.4 Heavy-Light Decomposition

```
// 2acdda
class HLD {
public:
    void init(int n) {
        p.resize(n);
        h.resize(n);
        in.resize(n);
        sz.resize(n);
        adj.resize(n);
        head.resize(n);
    }

    void addEdge(int a, int b) {
        adj[a].push_back(b);
        adj[b].push_back(a);
    }

    void build(int root = 0) {
        t = 0;
```

```
        head[root] = p[root] = root;
        buildChains(root, root);
        buildHld(root, root);
    }

private:
    vector<int> p, h, in, sz, head;
    vector<vector<int>> adj;
    int t;

    void buildChains(int on, int par) {
        sz[on] = 1;
        p[on] = par;
        for (auto &to : adj[on]) {
            if (to == par) {
                swap(to, adj[on].back());
                continue;
            }
            h[to] = h[on] + 1;
            buildChains(to, on);
            sz[on] += sz[to];
            if (sz[to] > sz[adj[on][0]]) {
                swap(to, adj[on][0]);
            }
        }
    }

    void buildHld(int on, int par) {
        in[on] = t++;
        for (auto to : adj[on]) {
            head[to] = (to == adj[on][0] ? head[on] : to);
            buildHld(to, on);
        }
    }
};
```

3.5 Min Cost Max Flow

```
// 6ece39
// by tfg50
// found at:
   https://github.com/tfg50/Competitive-Programming/blob/master/Bi
template <class T = int>
class MCMF {
public:
    struct Edge {
        Edge(int a, T b, T c) : to(a), cap(b), cost(c) {}
        int to;
        T cap, cost;
    };

    MCMF(int size) {
        n = size;
        edges.resize(n);
        pot.assign(n, 0);
        dist.resize(n);
        visit.assign(n, false);
    }

    std::pair<T, T> mcmf(int src, int sink) {
        std::pair<T, T> ans(0, 0);
        if(!SPFA(src, sink)) return ans;
        fixPot();
        // can use dijkstra to speed up depending on the
           graph
        while(SPFA(src, sink)) {
            auto flow = augment(src, sink);
            ans.first += flow.first;
            ans.second += flow.first * flow.second;
            fixPot();
        }
        return ans;
    }

    void addEdge(int from, int to, T cap, T cost) {
        edges[from].push_back(list.size());
        list.push_back(Edge(to, cap, cost));
        edges[to].push_back(list.size());
        list.push_back(Edge(from, 0, -cost));
    }
private:
```

```
int n;
std::vector<std::vector<int>> edges;
std::vector<Edge> list;
std::vector<int> from;
std::vector<T> dist, pot;
std::vector<bool> visit;

/*bool dij(int src, int sink) {
    T INF = std::numeric_limits<T>::max();
    dist.assign(n, INF);
    from.assign(n, -1);
    visit.assign(n, false);
    dist[src] = 0;
    for(int i = 0; i < n; i++) {
        int best = -1;
        for(int j = 0; j < n; j++) {
            if(visit[j]) continue;
            if(best == -1 || dist[best] > dist[j]) best
               = j;
        }
        if(dist[best] >= INF) break;
        visit[best] = true;
        for(auto e : edges[best]) {
            auto ed = list[e];
            if(ed.cap == 0) continue;
            T toDist = dist[best] + ed.cost + pot[best]
               - pot[ed.to];
            assert(toDist >= dist[best]);
            if(toDist < dist[ed.to]) {
                dist[ed.to] = toDist;
                from[ed.to] = e;
            }
        }
    }
    return dist[sink] < INF;
}*/

std::pair<T, T> augment(int src, int sink) {
    std::pair<T, T> flow = {list[from[sink]].cap, 0};
    for(int v = sink; v != src; v = list[from[v]^1].to) {
        flow.first = std::min(flow.first,
           list[from[v]].cap);
        flow.second += list[from[v]].cost;
    }
    for(int v = sink; v != src; v = list[from[v]^1].to) {
        list[from[v]].cap -= flow.first;
        list[from[v]^1].cap += flow.first;
    }
    return flow;
}

std::queue<int> q;
bool SPFA(int src, int sink) {
    T INF = std::numeric_limits<T>::max();
    dist.assign(n, INF);
    from.assign(n, -1);
    q.push(src);
    dist[src] = 0;
    while(!q.empty()) {
        int on = q.front();
        q.pop();
        visit[on] = false;
        for(auto e : edges[on]) {
            auto ed = list[e];
            if(ed.cap == 0) continue;
            T toDist = dist[on] + ed.cost + pot[on] -
               pot[ed.to];
            if(toDist < dist[ed.to]) {
                dist[ed.to] = toDist;
                from[ed.to] = e;
                if(!visit[ed.to]) {
                    visit[ed.to] = true;
                    q.push(ed.to);
                }
            }
        }
    }
    return dist[sink] < INF;
}

void fixPot() {
    T INF = std::numeric_limits<T>::max();
    for(int i = 0; i < n; i++) {
        if(dist[i] < INF) pot[i] += dist[i];
```

```
        }
    }
};
```

4 Matematica

4.1 Combinatoria

```
// fe3faa

// remember to change BOUNDS accordingly!!!

using T = Mint<>;

T fat[MAXN], inv[MAXN];

void setup() {
    fat[0] = inv[0] = 1;
    for (int i = 1; i < MAXN; i++) {
        fat[i] = fat[i - 1] * i;
    }
    inv[MAXN - 1] = fexp(fat[MAXN - 1], MOD - 2);
    for (int i = MAXN - 2; i >= 1; i--) {
        inv[i] = inv[i + 1] * (i + 1);
    }
}

// C(n, k) = n choose k, number of ways to choose a set of k
   elements from a set of n elements
T C(int n, int k) {
    if (n < k || k < 0)
        return T(0);
    return fat[n] * inv[k] * inv[n - k];
}

// C(n) = n-th Catalan number - number of valid parenthesis
   sequences of size 2 * n
T C(int n) {
    return C(2 * n, n) - C(2 * n, n + 1);
}
```

4.2 Division Trick

```
// 5bf9bf

// by tfg
// found at:
   https://github.com/tfg50/Competitive-Programming/blob/master/Bi
// O(sqrt(n))
for(int l = 1, r; l <= n; l = r + 1) {
    r = n / (n / l);
    // n / i has the same value for l <= i <= r
}
```

4.3 FFT

```
// chamar com vector<cplx> para FFT, ou vector<mint> para NTT
//
// O(n log(n))
// 64e51b

// copied from:
   https://github.com/brunomaletta/Biblioteca/blob/master/Codigo/M

template<typename T> void fft(vector<T> &a, bool f, int N,
   vector<int> &rev){
    for (int i = 0; i < N; i++)
        if (i < rev[i])
            swap(a[i], a[rev[i]]);
    int l, r, m;
    vector<T> roots(N);
    for (int n = 2; n <= N; n *= 2){
        T::fill_rt(f, n, N, roots);

        for (int pos = 0; pos < N; pos += n){
            l = pos+0, r = pos+n/2, m = 0;
            while (m < n/2){
```

```
                auto t = roots[m]*a[r];
                a[r] = a[l] - t;
                a[l] = a[l] + t;
                l++; r++; m++;
            }
        }
    }
    if (f) {
        auto invN = T(1)/N;
        for(int i = 0; i < N; i++) a[i] = a[i]*invN;
    }
}

template<typename T> vector<T> convolution(vector<T> &a,
    vector<T> &b) {
    vector<T> l(a.begin(), a.end());
    vector<T> r(b.begin(), b.end());
    int ln = l.size(), rn = r.size();
    int N = ln+rn-1;
    int n = 1, log_n = 0;
    while (n <= N) { n <<= 1; log_n++; }
    vector<int> rev(n);
    for (int i = 0; i < n; ++i){
        rev[i] = 0;
        for (int j = 0; j < log_n; ++j)
            if (i & (1<<j))
                rev[i] |= 1 << (log_n-1-j);
    }
    assert(N <= n);
    l.resize(n);
    r.resize(n);
    fft(l, false, n, rev);
    fft(r, false, n, rev);
    for (int i = 0; i < n; i++)
        l[i] *= r[i];
    fft(l, true, n, rev);
    return l;
}
```

4.4 Inteiro Modular

```
// a9a997
template <class T>
T fexp(T b, long long e) {
    T ans = T(1);
    for (; e > 0; e /= 2) {
        if (e & 1ll)
            ans *= b;
        b *= b;
    }
    return ans;
}

// maybe store factorial inverses to reduce division cost (?)
template <int mod = MOD>
struct Mint {
    int val;

    Mint(int x = 0): val(x < 0 ? x + mod : x) {}

    void operator += (Mint<mod> o) { *this = *this + o; }
    void operator -= (Mint<mod> o) { *this = *this - o; }
    void operator *= (Mint<mod> o) { *this = *this * o; }
    void operator /= (Mint<mod> o) { *this = *this / o; }
    Mint<mod> operator + (Mint<mod> o) { return val + o.val
       >= mod ? val + o.val - mod : val + o.val; }
    Mint<mod> operator - (Mint<mod> o) { return val - o.val
       < 0 ? val - o.val + mod : val - o.val; }
    Mint<mod> operator * (Mint<mod> o) { return (int) (1ll *
       val * o.val % mod); }
    Mint<mod> operator / (Mint<mod> o) { return *this *
       fexp(o, mod - 2); }

    friend ostream& operator << (ostream &os, const
       Mint<mod> &p) {
        return os << p.val;
    }

    friend istream& operator >> (istream &is, Mint<mod> &p) {
        return is >> p.val;
    }
};
```

4.5 Linear Sieve of Eratosthenes

```
// 9af074

// Can be used to calculate Multiplicative Functions, such
   as phi(n) (Mobius)

int phi[MAXN];
bool isPrime[MAXN];
vector<int> primes;

void sieve(int n = MAXN - 1) {
    for (int i = 1; i <= n; i++) {
        isPrime[i] = true;
        phi[i] = 1;
    }
    for (int i = 2; i <= n; i++) {
        if (isPrime[i]) {
            primes.push_back(i);
            phi[i] = -1;
        }
        for (auto p : primes) {
            if (1ll * p * i > n)
                break;
            isPrime[i * p] = false;
            if (i % p == 0) {
                phi[i * p] = 0;
                break;
            } else {
                phi[i * p] = -phi[i];
            }
        }
    }
}
```

4.6 Matrix

```
// 5cf9d9

// It's preferable to declare global matrices
// based on:
   https://github.com/tfg50/Competitive-Programming/blob/master/Bi
template <const int n, const int m, class T = Mint<>>
struct Matrix {
    T mat[n][m];

    Matrix(int d = 0) {
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < m; j++) {
                v[i][j] = T(0);
            }
            if (i < m)
                v[i][i] = T(d);
        }
    }

    template <int p>
    Matrix<n, p, T> operator * (const Matrix<m, p, T> &o) {
        Matrix<n, p, T> ans;
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < p; j++) {
                for (int k = 0; k < m; k++) {
                    ans.mat[i][j] = ans.mat[i][j] +
                       mat[i][k] * o.mat[k][j];
                }
            }
        }
        return ans;
    }
};
```

4.7 Miller Rabin e Pollard Rho

```
// 832086

// copied from:
   https://github.com/tfg50/Competitive-Programming/blob/master/Bi

//miller_rabin
typedef unsigned long long ull;
```

```
typedef long double ld;

ull fmul(ull a, ull b, ull m) {
    ull q = (ld) a * (ld) b / (ld) m;
    ull r = a * b - q * m;
    return (r + m) % m;
}

ull fexp(ull x, ull e, ull m) {
    ull ans = 1;
    x = x % m;
    for(; e; e >>= 1) {
        if(e & 1) {
            ans = fmul(ans, x, m);
        }
        x = fmul(x, x, m);
    }
    return ans;
}

bool miller(ull p, ull a) {
    ull s = p - 1;
    while(s % 2 == 0) s >>= 1;
    while(a >= p) a >>= 1;
    ull mod = fexp(a, s, p);
    while(s != p - 1 && mod != 1 && mod != p - 1) {
        mod = fmul(mod, mod, p);
        s <<= 1;
    }
    if(mod != p - 1 && s % 2 == 0)return false;
    else return true;
}

bool prime(ull p) {
    if(p <= 3)
        return true;
    if(p % 2 == 0)
        return false;
    return miller(p, 2) && miller(p, 3)
        && miller(p, 5) && miller(p, 7)
        && miller(p, 11) && miller(p, 13)
        && miller(p, 17) && miller(p, 19)
        && miller(p, 23) && miller(p, 29)
        && miller(p, 31) && miller(p, 37);
}

//pollard_rho

ull func(ull x, ull c, ull n) {
    return (fmul(x, x, n) + c) % n;
}

ull gcd(ull a, ull b) {
    if(!b) return a;
    else return gcd(b, a % b);
}

ull rho(ull n) {
    if(n % 2 == 0) return 2;
    if(prime(n)) return n;
    while(1) {
        ull c;
        do {
            c = rand() % n;
        } while(c == 0 || (c + 2) % n == 0);
        ull x = 2, y = 2, d = 1;
        ull pot = 1, lam = 1;
        do {
            if(pot == lam) {
                x = y;
                pot <<= 1;
                lam = 0;
            }
            y = func(y, c, n);
            lam++;
            d = gcd(x >= y ? x - y : y - x, n);
        } while(d == 1);
        if(d != n) return d;
    }
}

std::vector<ull> factors(ull n) {
    std::vector<ull> ans, rest, times;
    if(n == 1) return ans;
```

```
    rest.push_back(n);
    times.push_back(1);
    while(!rest.empty()) {
        ull x = rho(rest.back());
        if(x == rest.back()) {
            int freq = 0;
            for(int i = 0; i < rest.size(); i++) {
                int cur_freq = 0;
                while(rest[i] % x == 0) {
                    rest[i] /= x;
                    cur_freq++;
                }
                freq += cur_freq * times[i];
                if(rest[i] == 1) {
                    std::swap(rest[i], rest.back());
                    std::swap(times[i], times.back());
                    rest.pop_back();
                    times.pop_back();
                    i--;
                }
            }
            while(freq--) {
                ans.push_back(x);
            }
            continue;
        }
        ull e = 0;
        while(rest.back() % x == 0) {
            rest.back() /= x;
            e++;
        }
        e *= times.back();
        if(rest.back() == 1) {
            rest.pop_back();
            times.pop_back();
        }
        rest.push_back(x);
        times.push_back(e);
    }
    return ans;
}
```

5 Strings

5.1 Aho Corasick

```
// efdf19

// based on:
   https://github.com/tfg50/Competitive-Programming/blob/master/Bi
template <const int ALPHA = 26, class T = string, const int
   OFFSET = 'a'>
struct Aho {
    struct Node {
        int nxt[ALPHA];
        int size;
        int link, elink;
        bool end;

        Node() {
            for (int i = 0; i < ALPHA; i++) {
                nxt[i] = 0;
            }
            size = 0;
            link = elink = 0;
            end = false;
            // initialize new stuff here
        }
        // add new stuff here
    };

    vector<Node> nodes;

    Aho() {
        nodes.push_back(Node());
    }

    template <class F>
    void goUp(int on, F f) {
        for (on = nodes[on].end ? on : nodes[on].elink; on >
           0; on = nodes[on].elink) {
            f(nodes[on]);
        }
```

```
}

template <class C>
int nextState(int on, C c) const {
    return nodes[on].nxt[c - OFFSET];
}

int add(const T &s) {
    int cur = 0;
    for (auto ch : s) {
        if (nodes[cur].nxt[ch - OFFSET] == 0) {
            nodes[cur].nxt[ch - OFFSET] = (int)
               nodes.size();
            nodes.push_back(Node());
            nodes.back().size = nodes[cur].size + 1;
        }
        cur = nodes[cur].nxt[ch - OFFSET];
    }
    // mark this node as the end of a word
    nodes[cur].end = true;
    return cur;
}

void build() {
    queue<int> q;
    q.push(0);
    while (!q.empty()) {
        int cur = q.front();
        q.pop();
        nodes[cur].elink = (nodes[nodes[cur].link].end ?
           nodes[cur].link :
           nodes[nodes[cur].link].elink);
        for (int i = 0; i < ALPHA; i++) {
            int &nxt = nodes[cur].nxt[i];
            if (nxt) {
                nodes[nxt].link = (cur == 0 ? 0 :
                   nodes[nodes[cur].link].nxt[i]);
                q.push(nxt);
            } else {
                nxt = nodes[nodes[cur].link].nxt[i];
            }
        }
    }
};
```

5.2 Algoritmo Z

```
// d8a24c

// Classic problems using Zfunction:
// - String Matching Problem
// - Number of Different Substrings in O(n^2)
// - Find the root of a String
// - Search with at most 1 mistake in O(n)

template <class T>
struct ZFunc {
    // z[i] = lenght of the longest common preffix of v[0,
       n) and v[i, n)
    vector<int> z;
    ZFunc(const T &v): z((int) v.size()) {
        int n = (int) v.size(), l = 0, r = 0;
        if (!z.empty()) z[0] = n;
        for (int i = 1; i < n; i++) {
            if (i <= r) z[i] = min(z[i - l], r - i + 1);
            while (i + z[i] < n && v[i + z[i]] == v[z[i]])
               z[i]++;
            if (r < i + z[i] - 1) l = i, r = i + z[i] - 1;
        }
    }
};
```

5.3 Hash Polinomial

```
// c78010
struct Hash {
    const int p[2] = {337, 521}; // remember to change bases
       if needed
    const int m[2] = {(int) 1e9 + 7, (int) 1e9 + 9};
```

```
    int n;
    bool single;
    vector<int> h[2], pw[2];

    Hash(string &s, bool fs = false) {
        n = (int) s.size();
        single = fs;
        for (int k : {0, 1}) {
            h[k] = pw[k] = vector<int>(n);
            pw[k][0] = 1;
            h[k][0] = (int) s[0];
            for (int i = 1; i < n; i++) {
                h[k][i] = (1ll * h[k][i - 1]  * p[k] + 1ll *
                    (int) s[i]) % m[k];
                pw[k][i] = 1ll * pw[k][i - 1] * p[k] % m[k];
            }
            if (single) break;
        }
    }

    pair<int, int> substring(int l, int r) {
        if (l > r)
            swap(l, r);
        pair<int, int> ans = {0, 0};
        for (int k : {0, 1}) {
            int val = h[k][r] - (l > 0 ? 1ll * h[k][l - 1] *
                pw[k][r - l + 1] % m[k] : 0);
            val %= m[k];
            if (val < 0)
                val += m[k];
            if (!k) ans.first = val;
            else ans.second = val;
            if (single) break;
        }
        return ans;
    }
};
```

5.4 KMP - Knuth Morris Pratt

```
// ce4156
template <class T>
vector<int> prefixFunction(const T &v) {
    vector<int> p((int) v.size(), 0);
    for (int i = 1; i < (int) v.size(); i++) {
        p[i] = p[i - 1];
        while (p[i] > 0 && v[p[i]] != v[i]) p[i] = p[p[i] -
            1];
        if (v[p[i]] == v[i]) p[i]++;
    }
    return p;
}

// Assumes that there is a '#' at the end of pattern
template <class T, class F>
void match(const T &txt, const T &pat, const vector<int> &p,
    F f) {
    for (int i = 0, k = 0; i < (int) txt.size(); i++) {
        while (k > 0 && pat[k] != txt[i]) k = p[k - 1];
        if (pat[k] == txt[i]) k++;
        if (k + 1 == (int) pat.size()) {
            // a match was found!
            f(i - k + 1, i);
        }
    }
}
```

5.5 Manacher

```
// b4fb35

// copied from:
    https://github.com/tfg50/Competitive-Programming/blob/master/Bi
// manacher[0][i + 1] is the length of matches of even
    length palindrome, starting from [i, i + 1]
// manacher[1][i] is the length of matches of odd length
    palindrome, starting from [i, i]
std::array<std::vector<int>, 2> manacher(const std::string&
    s) {
    int n = (int) s.size();
```

```
    std::array<std::vector<int>, 2> p = {std::vector<int>(n
        + 1), std::vector<int>(n)};
    for(int z = 0; z < 2; z++) for (int i = 0, l = 0, r = 0;
        i < n; i++) {
        int t = r - i + !z;
        if (i < r) p[z][i] = std::min(t, p[z][l + t]);
        int L = i - p[z][i], R = i + p[z][i] - !z;
        while (L >= 1 && R + 1 < n && s[L - 1] == s[R + 1])
            p[z][i]++, L--, R++;
        if (R > r) l = L, r = R;
    }
    return p;
}
```

5.6 Suffix Array

```
// 675a66

// by tfg
class SuffixArray {
public:
    template<class T>
    std::vector<int> buildSuffix(const T &array) {
        int n = array.size();
        std::vector<int> sa(n);
        for(int i = 0; i < n; i++) {
            sa[i] = i;
        }
        std::sort(sa.begin(), sa.end(), [&](int a, int b) {
            return array[a] < array[b]; });
        int cur = 0;
        std::vector<int> inv(n);
        std::vector<int> nxt(n);
        inv[sa[0]] = 0;
        for(int i = 1; i < n; i++) {
            inv[sa[i]] = (array[sa[i - 1]] != array[sa[i]] ?
                ++cur : cur);
        }
        cur++;
        for(int k = 0; cur < n; k++) {
            cur = 0;
            auxSort(sa, inv, 1 << k);
            for(int l = 0, r = 0; l < n; l = r, cur++) {
                while(r < n && getPair(inv, sa[l], 1 << k)
                    == getPair(inv, sa[r], 1 << k)) {
                    nxt[sa[r++]] = cur;
                }
            }
            nxt.swap(inv);
        }
        return sa;
    }

    template<class T>
    std::vector<int> buildLCP(const std::vector<int> &sa,
        const T &array) {
        int n = sa.size();
        std::vector<int> inv(n);
        for(int i = 0; i < n; i++) {
            inv[sa[i]] = i;
        }
        std::vector<int> lcp(n, 0);
        for(int i = 0, k = 0; i < n; i++) {
            if(inv[i] + 1 == n) {
                k = 0;
                continue;
            }
            int j = sa[inv[i] + 1];
            while(i + k < n && j + k < n && array[i + k] ==
                array[j + k]) {
                k++;
            }
            lcp[inv[i]] = k;
            if(k > 0) {
                k--;
            }
        }
        return lcp;
    }
private:
    void auxSort(std::vector<int> &sa, const
        std::vector<int> &inv, int offset) {
```

```
        // O(nlogn) step, O(nlog^2n) total
        std::sort(sa.begin(), sa.end(), [&](int a, int b) {
            return getPair(inv, a, offset) < getPair(inv, b,
            offset); });
        // O(n) step, O(nlogn) total -- TO DO --
    }
    std::pair<int, int> getPair(const std::vector<int> &inv,
        int pos, int offset) {
        return std::pair<int, int>(inv[pos], pos + offset <
            (int) inv.size() ? inv[pos + offset] : -1);
    }
};
```

6 Extra

6.1 hash.sh

```
# Para usar (hash das linhas [l1, l2]):
# ./hash.sh arquivo.cpp l1 l2
# vlw Bruno :)
sed -n $2','$3' p' $1 | sed '/^#w/d' | cpp -dD -P
   -fpreprocessed | tr -d '[:space:]' | md5sum | cut -c-6
```

6.2 stress.sh

```
P=a
make ${P} ${P}2 gen || exit 1
for ((i = 1; ; i++)) do
    ./gen $i > in
    ./${P} < in > out
    ./${P}2 < in > out2
    if (! cmp -s out out2) then
        echo "--> entrada:"
        cat in
        echo "--> saida1:"
        cat out
        echo "--> saida2:"
        cat out2
        break;
    fi
    echo $i
done
```

6.3 template.cpp

```
#include <bits/stdc++.h>

using namespace std;
using ll = long long;
using pii = pair<int, int>;
```

```cpp
using pll = pair<ll, ll>;

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

const int MOD = 1e9 + 7;
const int MAXN = 2e5 + 5;
const ll INF = 1e18;

int main() {
    ios::sync_with_stdio(false);
    cin.tie(0);

    return 0;
}
```