BlackSackProblem [UFMG]

Bruno Monteiro, Pedro Papa e Rafael Grandsire

Índice

1 Estruturas

1.1 SegTree

```
// Query: soma do range [a, b]
// Update: soma x em cada elemento do range [a, b]
//
// Complexidades:
// build - O(n)
// query - O(log(n))
// update - O(log(n))

namespace seg {
    ll seg[4*MAX], lazy[4*MAX];
    int n, *v;

    ll build(int p=1, int l=0, int r=n-1) {
        lazy[p] = 0;
        if (l == r) return seg[p] = v[l];
        int m = (l+r)/2;
        return seg[p] = build(2*p, l, m) + build(2*p+1, m+1,
            r);
    }
    void build(int n2, int* v2) {
        n = n2, v = v2;
```

```
        build();
    }
    void prop(int p, int l, int r) {
        seg[p] += lazy[p]*(r-l+1);
        if (l != r) lazy[2*p] += lazy[p], lazy[2*p+1] +=
            lazy[p];
        lazy[p] = 0;
    }
    ll query(int a, int b, int p=1, int l=0, int r=n-1) {
        prop(p, l, r);
        if (a <= l and r <= b) return seg[p];
        if (b < l or r < a) return 0;
        int m = (l+r)/2;
        return query(a, b, 2*p, l, m) + query(a, b, 2*p+1,
            m+1, r);
    }
    ll update(int a, int b, int x, int p=1, int l=0, int
        r=n-1) {
        prop(p, l, r);
        if (a <= l and r <= b) {
            lazy[p] += x;
            prop(p, l, r);
            return seg[p];
        }
        if (b < l or r < a) return seg[p];
        int m = (l+r)/2;
        return seg[p] = update(a, b, x, 2*p, l, m) +
            update(a, b, x, 2*p+1, m+1, r);
    }
};
```

1.2 SegTree 2D

```
// Consultas 0-based
// Um valor inicial em (x, y) deve ser colocado em
    seg[x+n][y+n]
// Query: soma do retangulo ((x1, y1), (x2, y2))
// Update: muda o valor da posicao (x, y) para val
// Nao pergunte como que essa coisa funciona
//
// Para query com distancia de manhattan <= d, faca
// nx = x+y, ny = x-y
// Update em (nx, ny), query em ((nx-d, ny-d), (nx+d, ny+d))
//
// Complexidades:
// build - O(n^2)
// query - O(log^2(n))
// update - O(log^2(n))

int seg[2*MAX][2*MAX], n;

void build() {
    for (int x = 2*n; x; x--) for (int y = 2*n; y; y--) {
        if (x < n) seg[x][y] = seg[2*x][y] + seg[2*x+1][y];
        if (y < n) seg[x][y] = seg[x][2*y] + seg[x][2*y+1];
    }
}

int query(int x1, int y1, int x2, int y2) {
    int ret = 0, y3 = y1 + n, y4 = y2 + n;
    for (x1 += n, x2 += n; x1 <= x2; ++x1 /= 2, --x2 /= 2)
        for (y1 = y3, y2 = y4; y1 <= y2; ++y1 /= 2, --y2 /=
            2) {
            if (x1%2 == 1 and y1%2 == 1) ret += seg[x1][y1];
            if (x1%2 == 1 and y2%2 == 0) ret += seg[x1][y2];
            if (x2%2 == 0 and y1%2 == 1) ret += seg[x2][y1];
            if (x2%2 == 0 and y2%2 == 0) ret += seg[x2][y2];
        }

    return ret;
}

void update(int x, int y, int val) {
    int y2 = y += n;
    for (x += n; x; x /= 2, y = y2) {
        if (x >= n) seg[x][y] = val;
        else seg[x][y] = seg[2*x][y] + seg[2*x+1][y];

        while (y /= 2) seg[x][y] = seg[x][2*y] +
            seg[x][2*y+1];
    }
}
```

1.3 SegTree Esparça

```
// Query: soma do range [a, b]
// Update: flipa os valores de [a, b]
//
// Complexidades:
// build - O(n)
// query - O(log^2(n))
// update - O(log^2(n))

namespace seg {
    unordered_map<int, int> t;
    unordered_map<int, int> lazy;

    void build() { t.clear(), lazy.clear(); }

    void prop(int p, int l, int r) {
        if (!lazy[p]) return;
        t[p] = r-l+1-t[p];
        if (l != r) lazy[2*p]^=lazy[p], lazy[2*p+1]^=lazy[p];
        lazy[p] = 0;
    }

    int query(int a, int b, int p=1, int l=0, int r=N-1) {
        prop(p, l, r);
        if (b < l or r < a) return 0;
        if (a <= l and r <= b) return t[p];

        int m = l+r>>1;
        return query(a, b, 2*p, l, m)+query(a, b, 2*p+1,
           m+1, r);
    }

    int update(int a, int b, int p=1, int l=0, int r=N-1) {
        prop(p, l, r);
        if (b < l or r < a) return t[p];
        if (a <= l and r <= b) {
            lazy[p] ^= 1;
            prop(p, l, r);
            return t[p];
        }
        int m = l+r>>1;
        return t[p] = update(a, b, 2*p, l, m)+update(a, b,
           2*p+1, m+1, r);
    }
};
```

1.4 SegTree Iterativa

```
// Consultas 0-based
// Valores iniciais devem estar em (seg[n], ... , seg[2*n-1])
// Query: soma do range [a, b]
// Update: muda o valor da posicao p para x
//
// Complexidades:
// build - O(n)
// query - O(log(n))
// update - O(log(n))

int seg[2 * MAX];
int n;

void build() {
    for (int i = n - 1; i; i--) seg[i] = seg[2*i] +
       seg[2*i+1];
}

int query(int a, int b) {
    int ret = 0;
    for(a += n, b += n; a <= b; ++a /= 2, --b /= 2) {
        if (a % 2 == 1) ret += seg[a];
        if (b % 2 == 0) ret += seg[b];
    }
    return ret;
}

void update(int p, int x) {
    seg[p += n] = x;
    while (p /= 2) seg[p] = seg[2*p] + seg[2*p+1];
}
```

1.5 SegTree Iterativa com Lazy

```
// SegTree 1-based
// Valores iniciais devem estar em (seg[n], ... , seg[2*n-1])
// Query: soma do range [a, b], 0-based
// Update: soma x em cada elemento do range [a, b], 0-based
//
// Complexidades:
// build - O(n)
// query - O(log(n))
// update - O(log(n))

int seg[2*MAX];
int lazy[2*MAX];
int n;

void build() {
    for (int i = n - 1; i; i--) seg[i] = seg[2*i] +
        seg[2*i+1];
    memset(lazy, 0, sizeof(lazy));
}

// soma x na posicao p de tamanho tam
void poe(int p, int x, int tam) {
    seg[p] += x * tam;
    if (p < n) lazy[p] += x;
}

// atualiza todos os pais da folha p
void sobe(int p) {
    for (int tam = 2; p /= 2; tam *= 2)
        seg[p] = seg[2*p] + seg[2*p+1] + lazy[p] * tam;
}

// propaga o caminho da raiz ate a folha p
void prop(int p) {
    int tam = 1 << 29;
    for (int s = 30; s; s--, tam /= 2) {
        int i = p >> s;
        if (lazy[i]) {
            poe(2*i, lazy[i], tam);
            poe(2*i+1, lazy[i], tam);
            lazy[i] = 0;
        }
    }
}

int query(int a, int b) {
    prop(a += n), prop(b += n);
    int ret = 0;
    for(; a <= b; a /= 2, b /= 2) {
        if (a % 2 == 1) ret += seg[a++];
        if (b % 2 == 0) ret += seg[b--];
    }
    return ret;
}

void update(int a, int b, int x) {
    int a2 = a += n, b2 = b += n, tam = 1;
    for (; a <= b; a /= 2, b /= 2, tam *= 2) {
        if (a % 2 == 1) poe(a++, x, tam);
        if (b % 2 == 0) poe(b--, x, tam);
    }
    sobe(a2), sobe(b2);
}
```

1.6 BIT

```
// BIT 1-based, v 0-based
// Para mudar o valor da posicao p para x,
// faca: poe(x - query(p, p), p)
// l_bound(x) retorna o menor p tal que
// query(1, p+1) > x    (0 based!)
//
// Complexidades:
// build - O(n)
// poe - O(log(n))
// query - O(log(n))
// l_bound - O(log(n))

int n;
int bit[MAX];
int v[MAX];

void build() {
    bit[0] = 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];
    }
}

// soma x na posicao p
void poe(int x, int p) {
    for (; p <= n; p += p & -p) bit[p] += x;
}

// soma [1, p]
int pref(int p) {
    int ret = 0;
    for (; p; p -= p & -p) ret += bit[p];
    return ret;
}

// soma [a, b]
int query(int a, int b) {
    return query(b) - query(a - 1);
}

int l_bound(ll x) {
    int p = 0;
    for (int i = MAX2; i+1; i--) if (p + (1<<i) <= n
        and bit[p + (1<<i)] <= x) x -= bit[p += (1<<i)];
    return p;
}
```

1.7 BIT 2D

```
// BIT 1-based
// Para mudar o valor da posicao (x, y) para k,
// faca: poe(x, y, k - sum(x, y, x, y))
//
// Complexidades:
// poe - O(log^2(n))
// query - O(log^2(n))

int n;
int bit[MAX][MAX];

void poe(int x, int y, int k) {
    for (int y2 = y; x <= n; x += x & -x)
        for (y = y2; y <= n; y += y & -y)
            bit[x][y] += k;
}

int sum(int x, int y) {
    int ret = 0;
    for (int y2 = y; x; x -= x & -x)
        for (y = y2; y; y -= y & -y)
            ret += bit[x][y];

    return ret;
}

int query(int x, int y, int z, int w) {
    return sum(z, w) - sum(x-1, w)
        - sum(z, y-1) + sum(x-1, y-1);
}
```

1.8 DSU Persistente

```
// Complexidades:
// build - O(n)
// find - O(log(n))
// une - O(log(n))

int n, p[MAX], sz[MAX], ti[MAX];

void build() {
    for (int i = 0; i < n; i++) {
        p[i] = i;
        sz[i] = 1;
        ti[i] = -INF;
    }
}

int find(int k, int t) {
    if (p[k] == k or ti[k] > t) return k;
```

```
    return find(p[k], t);
}

void une(int a, int b, int t) {
    a = find(a); b = find(b);
    if (a == b) return;
    if (sz[a] > sz[b]) swap(a, b);

    sz[b] += sz[a];
    p[a] = b;
    ti[a] = t;
}
```

1.9 Mergesort Tree

```
// query(a, b, val) retorna numero de
// elementos em [a, b] <= val
// Usa O(n log(n)) de memoria
//
// Complexidades:
// build - O(n log(n))
// query - O(log^2(n))

#define ALL(x) x.begin(),x.end()

int v[MAX], n;
vector<vector<int> > tree(4*MAX);

void build(int p, int l, int r) {
    if (l == r) return tree[p].push_back(cr[l]);
    int m = (l+r)/2;
    build(2*p, l, m), build(2*p+1, m+1, r);
    merge(ALL(tree[2*p]), ALL(tree[2*p+1]),
        back_inserter(tree[p]));
}

int query(int a, int b, int val, int p=1, int l=0, int
   r=n-1) {
    if (b < l or r < a) return 0; // to fora
    if (a <= l and r <= b) // to totalmente dentro
        return lower_bound(ALL(tree[p]), val+1) -
            tree[p].begin();
    int m = (l+r)/2;
    return query(a, b, val, 2*p, l, m) + query(a, b, val,
        2*p+1, m+1, r);
}
```

1.10 Order Statistic Set

```
// Funciona do C++11 pra cima

#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
using namespace __gnu_pbds;
template <class T>
    using ord_set = tree<T, null_type, less<T>, rb_tree_tag,
    tree_order_statistics_node_update>;

// para declarar:
ord_set<int> s;
// coisas do set normal funcionam:
for (auto i : s) cout << i << endl;
cout << s.size() << endl;
// k-esimo maior elemento O(log|s|):
// k=0: menor elemento
cout << *s.find_by_order(k) << endl;
// quantos sao menores do que k O(log|s|):
cout << s.order_of_key(k) << endl;

// Para fazer um multiset, tem que
// usar ord_set<pair<int, int> > com o
// segundo parametro sendo algo para diferenciar
// os ementos iguais.
// s.order_of_key({k, -INF}) vai retornar o
// numero de elementos < k
```

1.11 Sparse-Table

```
// MAX2 = log(MAX)
//
// Complexidades:
// build - O(n log(n))
// query - O(1)
```

```
int n;
int v[MAX];
int m[MAX][MAX2]; // m[i][j] : posicao do minimo
                  // em [v[i], v[i + 2^j - 1]]

void build() {
    for (int i = 0; i < n; i++) m[i][0] = i;

    for (int j = 1; 1 << j <= n; j++) {
        int tam = 1 << j;
        for (int i = 0; i + tam <= n; i++) {
            if (v[m[i][j - 1]] < v[m[i + tam/2][j - 1]])
                m[i][j] = m[i][j - 1];
            else m[i][j] = m[i + tam/2][j - 1];
        }
    }
}

int query(int a, int b) {
    int j = (int) log2(b - a + 1);

    return min(v[m[a][j]], v[m[b - (1 << j) + 1][j]]);
}
```

1.12 SQRT-decomposition

```
// 0-indexed
// MAX2 = sqrt(MAX)
//
// O bloco da posicao x eh
// sempre x/q
//
// Complexidades:
// build - O(n)
// query - O(sqrt(n))

int n, q;
int v[MAX];
int bl[MAX2];

void build() {
    q = (int) sqrt(n);

     // computa cada bloco
    for (int i = 0; i <= q; i++) {
        bl[i] = INF;
        for (int j = 0; j < q and q * i + j < n; j++)
            bl[i] = min(bl[i], v[q * i + j]);
    }
}

int query(int a, int b) {
    int ret = INF;

    // linear no bloco de a
    for (; a <= b and a % q; a++) ret = min(ret, v[a]);

    // bloco por bloco
    for (; a + q <= b; a += q) ret = min(ret, bl[a / q]);

    // linear no bloco de b
    for (; a <= b; a++) ret = min(ret, v[a]);

    return ret;
}
```

1.13 Treap

```
// Complexidades:
//
// insert - O(log(n))
// erase - O(log(n))
// query - O(log(n))

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

template<typename T> struct treap {
    struct node {
        int p;
        int l, r;
        T v;
        int sz;
```

```
    T min_s;
    node(){}
    node(T v):p(rng()), l(-1), r(-1), v(v){}
} t[MAX];
int size(int i){
    if (i == -1) return 0;
    return t[i].sz;
}
void update(int i){
    if (i == -1) return;
    t[i].min_s = t[i].v;

    int l = t[i].l;
    int r = t[i].r;
    t[i].sz = 1 + size(l) + size(r);

    if (l != -1)
        t[i].min_s = min(t[i].min_s, t[l].min_s);
    if (r != -1)
        t[i].min_s = min(t[i].min_s, t[r].min_s);
}
void split(int i, int k, int &l, int &r){ //key
    if (i == -1){
        l = -1; r = -1;
        return;
    }
    if (t[i].v < k){
        split(t[i].r, k, l, r);
        t[i].r = l;
        l = i;
    }
    else{
        split(t[i].l, k, l, r);
        t[i].l = r;
        r = i;
    }
    update(i);
}
void split_implicit(int i, int k, int &l, int &r, int sz
   = 0){
    if (i == -1){
        l = -1; r = -1;
        return;
    }
    int inc = size(t[i].l); //quantidade elementos menor
       que k
    if (sz+inc < k){
        split_implicit(t[i].r, k, l, r, sz+inc+1);
        t[i].r = l;
        l = i;
    }
    else{
        split_implicit(t[i].l, k, l, r, sz);
        t[i].l = r;
        r = i;
    }
    update(i);
}
int merge(int l, int r){ //priority
    if (l == -1) {
        update(r);
        return r;
    }
    if (r == -1) {
        update(l);
        return l;
    }
    if (t[l].p > t[r].p){
        t[l].r = merge(t[l].r, r);
        update(l);
        return l;
    }
    else{
        t[r].l = merge(l, t[r].l);
        update(r);
        return r;
    }
}

int it = 0;
void insert(int &root, T v){
    int M = it++;
    t[M] = node(v);
    if (root == -1) {
```

```
            root = M;
            return;
        }
        root = merge(root, M);
    }

    T query(int &root, int L, int R){
        int l, m, r;
        split_implicit(root, R+1, m, r);
        split_implicit(m, L, l, m);

        T ans = t[m].min_s;
        l = merge(l, m);
        l = merge(l, r);
        root = l;
        return ans;
    }
    void erase(int &root, int pos){
        int l, m, r;
        split_implicit(root, pos+1, m, r);
        split_implicit(m, pos, l, m);
        l = merge(l, r);
        root = l;
    }
};
```

1.14 Trie

```
// N deve ser maior ou igual ao numero de nos da trie
// fim indica se alguma palavra acaba nesse no
//
// Complexidade:
// Inserir e conferir string S -> O(|S|)

// usar static trie T
// T.insert(s) para inserir
// T.find(s) para ver se ta
// T.prefix(s) printa as strings
// que tem s como prefixo

struct trie{
    map<char, int> t[MAX+5];
    int p;
    trie(){
        p = 1;
    }
    void insert(string s){
        s += '$';
        int i = 0;
        for (char c : s){
            auto it = t[i].find(c);
            if (it == t[i].end())
                i = t[i][c] = p++;
            else
                i = it->second;
        }
    }
    bool find(string s){
        s += '$';
        int i = 0;
        for (char c : s){
            auto it = t[i].find(c);
            if (it == t[i].end()) return false;
            i = it->second;
        }
        return true;
    }
    void prefix(string &l, int i){
        if (t[i].find('$') != t[i].end())
            cout << "  " << l << endl;
        for (auto p : t[i]){
            l += p.first;
            prefix(l, p.second, k);
            l.pop_back();
        }
    }
    void prefix(string s){
        int i = 0;
        for (char c : s){
            auto it = t[i].find(c);
            if (it == t[i].end()) return;
            i = it->second;
        }
        int k = 0;
```

```
        prefix(s, i, k);
    }
};
```

1.15 Wavelet-Tree

```
// Usa O(sigma + n log(sigma)) de memoria,
// onde sigma = MAXN - MINN
// query(i, j) retorna o numero de elementos de
// v[i, j) que pertencem a [x, y]
// kth(i, j, k) retorna o elemento que estaria
// na poscicao k-1 de v[i, j), se ele fosse ordenado
//
// Complexidades:
// build - O(n log(sigma))
// query - O(log(sigma))
// kth   - O(log(sigma))

int n, v[MAX], x, y;
vector<vector<int> > esq(4*(MAXN-MINN));

void build(int b = 0, int e = n, int p = 1, int l = MINN,
    int r = MAXN) {
     if (l == r) return;
     int m = (l+r)/2; esq[p].push_back(0);
     for (int i = b; i < e; i++)
        esq[p].push_back(esq[p].back()+(v[i]<=m));

     int m2 = stable_partition(v+b, v+e, [=](int i){return i
        <= m;}) - v;
     build(b, m2, 2*p, l, m), build(m2, e, 2*p+1, m+1, r);
}

int query(int i, int j, int p = 1, int l = MINN, int r =
    MAXN) {
     if (y < l or r < x) return 0;
     if (x <= l and r <= y) return j-i;
     int m = (l+r)/2, ei = esq[p][i], ej = esq[p][j];
     return query(ei, ej, 2*p, l, m) + query(i-ei, j-ej,
        2*p+1, m+1, r);
}

int kth(int i, int j, int k, int p=1, int l = MINN, int r =
    MAXN) {
     if (l == r) return l;
     int m = (l+r)/2, ei = esq[p][i], ej = esq[p][j];
     if (k <= ej-ei) return kth(ei, ej, k, 2*p, l, m);
     return kth(i-ei, j-ej, k-(ej-ei), 2*p+1, m+1, r);
}
```

2 Grafos

2.1 2-SAT

```
// Retorna se eh possivel atribuir valores
// Grafo tem que caber 2n vertices
// add(x, y) adiciona implicacao x -> y
// Para adicionar uma clausula (x ou y)
// chamar add(nao(x), y)
// Se x tem que ser verdadeiro, chamar add(nao(x), x)
// O tarjan deve computar o componente conexo
// de cada vertice em comp
//
// O(|V|+|E|)

vector<vector<int> > g(MAX);
int n;

int nao(int x){ return (x + n) % (2*n); }

// x -> y  =  !x ou y
void add(int x, int y){
    g[x].pb(y);
    // contraposicao
    g[nao(y)].pb(nao(x));
}

bool doisSAT(){
    tarjan();
    for (int i = 0; i < m; i++)
        if (comp[i] == comp[nao(i)]) return 0;
    return 1;
```

```
}
```

2.2 Bellman-Ford

```
// Calcula a menor distancia
// entre a e todos os vertices e
// detecta ciclo negativo
// Retorna 1 se ha ciclo negativo
// Nao precisa representar o grafo,
// soh armazenar as arestas
//
// O(nm)

int n, m;
int d[MAX];
vector<pair<int, int> > ar; // vetor de arestas
vector<int> w;              // peso das arestas

bool bellman_ford(int a) {
    for (int i = 0; i < n; i++) d[i] = INF;
    d[a] = 0;

    for (int i = 0; i <= n; i++)
        for (int j = 0; j < m; j++) {
            if (d[ar[j].second] > d[ar[j].first] + w[j]) {
                if (i == n) return 1;

                d[ar[j].second] = d[ar[j].first] + w[j];
            }
        }

    return 0;
}
```

2.3 Floyd-Warshall

```
// encontra o menor caminho entre todo
// par de vertices e detecta ciclo negativo
// returna 1 sse ha ciclo negativo
// d[i][i] deve ser 0
// para i != j, d[i][j] deve ser w se ha uma aresta
// (i, j) de peso w, INF caso contrario
//
// O(n^3)

int n;
int d[MAX][MAX];

bool floyd_warshall() {
    for (int k = 0; k < n; k++)
    for (int i = 0; i < n; i++)
    for (int j = 0; j < n; j++)
        d[i][j] = min(d[i][j], d[i][k] + d[k][j]);

    for (int i = 0; i < n; i++)
        if (d[i][i] < 0) return 1;

    return 0;
}
```

2.4 Heavy-Light Decomposition Vértice

```
// SegTree de soma
// query / update de soma dos vertices
//
// Complexidades:
// build - O(n)
// query_path - O(log^2 (n))
// update_path - O(log^2 (n))
// query_subtree - O(log(n))
// update_subtree - O(log(n))

#define f first
#define s second

namespace seg {
    ll seg[4*MAX], lazy[4*MAX];
    int n, *v;

    ll build(int p=1, int l=0, int r=n-1) {
        lazy[p] = 0;
        if (l == r) return seg[p] = v[l];
        int m = (l+r)/2;
```

```cpp
        return seg[p] = build(2*p, l, m) + build(2*p+1, m+1,
           r);
    }
    void build(int n2, int* v2) {
        n = n2, v = v2;
        build();
    }
    void prop(int p, int l, int r) {
        seg[p] += lazy[p]*(r-l+1);
        if (l != r) lazy[2*p] += lazy[p], lazy[2*p+1] +=
           lazy[p];
        lazy[p] = 0;
    }
    ll query(int a, int b, int p=1, int l=0, int r=n-1) {
        prop(p, l, r);
        if (a <= l and r <= b) return seg[p];
        if (b < l or r < a) return 0;
        int m = (l+r)/2;
        return query(a, b, 2*p, l, m) + query(a, b, 2*p+1,
           m+1, r);
    }
    ll update(int a, int b, int x, int p=1, int l=0, int
       r=n-1) {
        prop(p, l, r);
        if (a <= l and r <= b) {
            lazy[p] += x;
            prop(p, l, r);
            return seg[p];
        }
        if (b < l or r < a) return seg[p];
        int m = (l+r)/2;
        return seg[p] = update(a, b, x, 2*p, l, m) +
            update(a, b, x, 2*p+1, m+1, r);
    }
};

namespace hld {
    vector<pair<int, int> > g[MAX];
    int in[MAX], out[MAX], sz[MAX];
    int peso[MAX], pai[MAX];
    int h[MAX], v[MAX], t;

    void build_hld(int k, int p = -1, int f = 1) {
        v[in[k] = t++] = peso[k]; sz[k] = 1;
        for (auto& i : g[k]) if (i.f != p) {
            pai[i.f] = k;
            h[i.f] = (i == g[k][0] ? h[k] : i.f);
            build_hld(i.f, k, f); sz[k] += sz[i.f];

            if (sz[i.f] > sz[g[k][0].f]) swap(i, g[k][0]);
        }
        out[k] = t;
        if (p*f == -1) build_hld(h[k] = k, -1, t = 0);
    }
    void build(int root = 0) {
        t = 0;
        build_hld(root);
        seg::build(t, v);
    }
    ll query_path(int a, int b) {
        if (a == b) return seg::query(in[a], in[a]);
        if (in[a] < in[b]) swap(a, b);

        if (h[a] == h[b]) return seg::query(in[b], in[a]);
        return seg::query(in[h[a]], in[a]) +
           query_path(pai[h[a]], b);
    }
    void update_path(int a, int b, int x) {
        if (a == b) return (void)seg::update(in[a], in[a],
           x);
        if (in[a] < in[b]) swap(a, b);

        if (h[a] == h[b]) return (void)seg::update(in[b],
           in[a], x);
        seg::update(in[h[a]], in[a], x);
           update_path(pai[h[a]], b, x);
    }
    ll query_subtree(int a) {
        if (in[a] == out[a]-1) return seg::query(in[a],
           in[a]);
        return seg::query(in[a], out[a]-1);
    }
    void update_subtree(int a, int x) {
        if (in[a] == out[a]-1) return
```

```
        (void)seg::update(in[a], in[a], x);
        seg::update(in[a], out[a]-1, x);
    }
    int lca(int a, int b) {
        if (in[a] < in[b]) swap(a, b);
        return h[a] == h[b] ? b : lca(pai[h[a]], b);
    }
};
```

2.5 Heavy-Light Decomposition Aresta

```
// SegTree de soma
// query / update de soma das arestas
//
// Complexidades:
// build - O(n)
// query_path - O(log^2 (n))
// update_path - O(log^2 (n))
// query_subtree - O(log(n))
// update_subtree - O(log(n))

#define f first
#define s second

namespace seg {
    ll seg[4*MAX], lazy[4*MAX];
    int n, *v;

    ll build(int p=1, int l=0, int r=n-1) {
        lazy[p] = 0;
        if (l == r) return seg[p] = v[l];
        int m = (l+r)/2;
        return seg[p] = build(2*p, l, m) + build(2*p+1, m+1,
            r);
    }
    void build(int n2, int* v2) {
        n = n2, v = v2;
        build();
    }
    void prop(int p, int l, int r) {
        seg[p] += lazy[p]*(r-l+1);
        if (l != r) lazy[2*p] += lazy[p], lazy[2*p+1] +=
            lazy[p];
        lazy[p] = 0;
    }
    ll query(int a, int b, int p=1, int l=0, int r=n-1) {
        prop(p, l, r);
        if (a <= l and r <= b) return seg[p];
        if (b < l or r < a) return 0;
        int m = (l+r)/2;
        return query(a, b, 2*p, l, m) + query(a, b, 2*p+1,
            m+1, r);
    }
    ll update(int a, int b, int x, int p=1, int l=0, int
        r=n-1) {
        prop(p, l, r);
        if (a <= l and r <= b) {
            lazy[p] += x;
            prop(p, l, r);
            return seg[p];
        }
        if (b < l or r < a) return seg[p];
        int m = (l+r)/2;
        return seg[p] = update(a, b, x, 2*p, l, m) +
            update(a, b, x, 2*p+1, m+1, r);
    }
};

namespace hld {
    vector<pair<int, int> > g[MAX];
    int in[MAX], out[MAX], sz[MAX];
    int sobe[MAX], pai[MAX];
    int h[MAX], v[MAX], t;

    void build_hld(int k, int p = -1, int f = 1) {
        v[in[k] = t++] = sobe[k]; sz[k] = 1;
        for (auto& i : g[k]) if (i.f != p) {
            sobe[i.f] = i.s; pai[i.f] = k;
            h[i.f] = (i == g[k][0] ? h[k] : i.f);
            build_hld(i.f, k, f); sz[k] += sz[i.f];

            if (sz[i.f] > sz[g[k][0].f]) swap(i, g[k][0]);
        }
        out[k] = t;
```

```
        if (p*f == -1) build_hld(h[k] = k, -1, t = 0);
    }
    void build(int root = 0) {
        t = 0;
        build_hld(root);
        seg::build(t, v);
    }
    ll query_path(int a, int b) {
        if (a == b) return 0;
        if (in[a] < in[b]) swap(a, b);

        if (h[a] == h[b]) return seg::query(in[b]+1, in[a]);
        return seg::query(in[h[a]], in[a]) +
           query_path(pai[h[a]], b);
    }
    void update_path(int a, int b, int x) {
        if (a == b) return;
        if (in[a] < in[b]) swap(a, b);

        if (h[a] == h[b]) return (void)seg::update(in[b]+1,
           in[a], x);
        seg::update(in[h[a]], in[a], x);
           update_path(pai[h[a]], b, x);
    }
    ll query_subtree(int a) {
        if (in[a] == out[a]-1) return 0;
        return seg::query(in[a]+1, out[a]-1);
    }
    void update_subtree(int a, int x) {
        if (in[a] == out[a]-1) return;
        seg::update(in[a]+1, out[a]-1, x);
    }
    int lca(int a, int b) {
        if (in[a] < in[b]) swap(a, b);
        return h[a] == h[b] ? b : lca(pai[h[a]], b);
    }
};
```

2.6 LCA

```
// Assume que um vertice eh ancestral dele mesmo, ou seja,
// se a eh ancestral de b, lca(a, b) = a
// MAX2 = ceil(log(MAX))
//
// Complexidades:
// build - O(n log(n))
// lca - O(log(n))

vector<vector<int> > g(MAX);
int n, p;
int pai[MAX2][MAX];
int in[MAX], out[MAX];

void dfs(int k) {
    in[k] = p++;
    for (int i = 0; i < (int) g[k].size(); i++)
        if (in[g[k][i]] == -1) {
            pai[0][g[k][i]] = k;
            dfs(g[k][i]);
        }
    out[k] = p++;
}

void build(int raiz) {
    for (int i = 0; i < n; i++) pai[0][i] = i;
    p = 0, memset(in, -1, sizeof in);
    dfs(raiz);

    // pd dos pais
    for (int k = 1; k < MAX2; k++) for (int i = 0; i < n;
       i++)
        pai[k][i] = pai[k - 1][pai[k - 1][i]];
}

bool anc(int a, int b) { // se a eh ancestral de b
    return in[a] <= in[b] and out[a] >= out[b];
}

int lca(int a, int b) {
    if (anc(a, b)) return a;
    if (anc(b, a)) return b;

    // sobe a
    for (int k = MAX2 - 1; k >= 0; k--)
```

```
        if (!anc(pai[k][a], b)) a = pai[k][a];

    return pai[0][a];
}
```

2.7 LCA com HLD

```
// Assume que um vertice eh ancestral dele mesmo, ou seja,
// se a eh ancestral de b, lca(a, b) = a
// Para buildar pasta chamar build(root)
//
// Complexidades:
// build - O(n)
// lca - O(log(n))

vector<vector<int> > g(MAX);
int in[MAX], h[MAX], sz[MAX];
int pai[MAX], t;

void build(int k, int p = -1, int f = 1) {
    in[k] = t++; sz[k] = 1;
    for (int& i : g[k]) if (i != p) {
        pai[i] = k;
        h[i] = (i == g[k][0] ? h[k] : i);
        build(i, k, f); sz[k] += sz[i];

        if (sz[i] > sz[g[k][0]]) swap(i, g[k][0]);
    }
    if (p*f == -1) t = 0, h[k] = k, build(k, -1, 0);
}

int lca(int a, int b) {
    if (in[a] < in[b]) swap(a, b);
    return h[a] == h[b] ? b : lca(pai[h[a]], b);
}
```

2.8 LCA com RMQ

```
// Assume que um vertice eh ancestral dele mesmo, ou seja,
// se a eh ancestral de b, lca(a, b) = a
//
// Complexidades:
// build - O(n) + build_RMQ
// lca - RMQ

int n;
vector<vector<int> > g(MAX);
int pos[MAX];     // pos[i] : posicao de i em v (primeira
   aparicao
int ord[2 * MAX]; // ord[i] : i-esimo vertice na ordem de
   visitacao da dfs
int v[2 * MAX];   // vetor de alturas que eh usado na RMQ
int p;

void dfs(int k, int l) {
    ord[p] = k;
    pos[k] = p;
    v[p++] = l;
    for (int i = 0; i < (int) g[k].size(); i++)
        if (pos[g[k][i]] == -1) {
            dfs(g[k][i], l + 1);
            ord[p] = k;
            v[p++] = l;
        }
}

void build(int root) {
    for (int i = 0; i < n; i++) pos[i] = -1;

    p = 0;
    dfs(root, 0);

    build_RMQ();
}

int lca(int u, int v) {
    int a = pos[u], b = pos[v];
    if (a > b) swap(a, b);
    return ord[RMQ(a, b)];
}
```

2.9 Tree Center

```
// Centro eh o vertice que minimiza
// a maior distancia dele pra alguem
// O centro fica no meio do diametro
// A funcao center retorna um par com
// o diametro e o centro
//
// O(n+m)

vector<vector<int> > g(MAX);
int n, vis[MAX];
int d[2][MAX];

// retorna ultimo vertice visitado
int bfs(int k, int x) {
        queue<int> q; q.push(k);
    memset(vis, 0, sizeof(vis));
    vis[k] = 1;
    d[x][k] = 0;
    int last = k;

    while (q.size()) {
        int u = q.front(); q.pop();
        last = u;
        for (int i : g[u]) if (!vis[i]) {
            vis[i] = 1;
            q.push(i);
            d[x][i] = d[x][u] + 1;
        }
    }
    return last;
}

pair<int, int> center() {
    int a = bfs(0, 0);
    int b = bfs(a, 1);
    bfs(b, 0);
    int c, mi = INF;
    for (int i = 0; i < n; i++) if (max(d[0][i], d[1][i]) <
        mi) {
        mi = max(d[0][i], d[1][i]), c = i;
    return {d[0][a], c};
}
```

2.10 Centroid decomposition

```
// O(n log(n))

int n;
vector<vector<int> > g(MAX);
int subsize[MAX];
int rem[MAX];
int pai[MAX];

void dfs(int k, int last) {
    subsize[k] = 1;
    for (int i = 0; i < (int) g[k].size(); i++)
        if (g[k][i] != last and !rem[g[k][i]]) {
            dfs(g[k][i], k);
            subsize[k] += subsize[g[k][i]];
        }
}

int centroid(int k, int last, int size) {
    for (int i = 0; i < (int) g[k].size(); i++) {
        int u = g[k][i];
        if (rem[u] or u == last) continue;
        if (subsize[u] > size / 2)
            return centroid(u, k, size);
    }
    // k eh o centroid
    return k;
}

void decomp(int k, int last) {
    dfs(k, k);

    // acha e tira o centroid
    int c = centroid(k, k, subsize[k]);
    rem[c] = 1;
    pai[c] = last;
    if (k == last) pai[c] = c;

    // decompoe as sub-arvores
    for (int i = 0; i < (int) g[c].size(); i++)
        if (!rem[g[c][i]]) decomp(g[c][i], c);
```

```
}

void build() {
    memset(rem, 0, sizeof rem);
    decomp(0, 0);
}
```

2.11 Dijkstra

```
// encontra menor distancia de a
// para todos os vertices
// se ao final do algoritmo d[i] = INF,
// entao a nao alcanca i
//
// O(m log(n))

int n;
vector<vector<int> > g(MAX);
vector<vector<int> > w(MAX); // peso das arestas
int d[MAX];

void dijsktra(int a) {
    for (int i = 0; i < n; i++) d[i] = INF;
    d[a] = 0;
    priority_queue<pair<int, int> > Q;
    Q.push(make_pair(0, a));

    while (Q.size()) {
        int u = Q.top().second, dist = -Q.top().first;
        Q.pop();
        if (dist > d[u]) continue;

        for (int i = 0; i < (int) g[u].size(); i++) {
            int v = g[u][i];
            if (d[v] > d[u] + w[u][i]) {
                d[v] = d[u] + w[u][i];
                Q.push(make_pair(-d[v], v));
            }
        }
    }
}
```

2.12 Dinic Dilson

```
// O(n^2 m)
// Grafo bipartido -> O(sqrt(n)*m)

template <class T> struct dinic {
    struct edge {
        int v, rev;
        T cap;
        edge(int v_, T cap_, int rev_) : v(v_), cap(cap_),
            rev(rev_) {}
    };
    vector<vector<edge>> g;
    vector<int> level;
    queue<int> q;
    T flow;
    int n;
    dinic(int n_) : g(n_), level(n_), n(n_) {}
    void add_edge(int u, int v, T cap) {
        if (u == v)
            return;
        edge e(v, cap, int(g[v].size()));
        edge r(u, 0, int(g[u].size()));
        g[u].push_back(e);
        g[v].push_back(r);
    }

    bool build_level_graph(int src, int sink) {
        fill(level.begin(), level.end(), -1);
        while (not q.empty())
            q.pop();
        level[src] = 0;
        q.push(src);
        while (not q.empty()) {
            int u = q.front();
            q.pop();
            for (auto e = g[u].begin(); e != g[u].end();
                ++e) {
                if (not e->cap or level[e->v] != -1)
                    continue;
                level[e->v] = level[u] + 1;
                if (e->v == sink)
```

```
                return true;
            q.push(e->v);
        }
    }
    return false;
}

T blocking_flow(int u, int sink, T f) {
    if (u == sink or not f)
        return f;
    T fu = f;
    for (auto e = g[u].begin(); e != g[u].end(); ++e) {
        if (not e->cap or level[e->v] != level[u] + 1)
            continue;
        T mincap = blocking_flow(e->v, sink, min(fu,
            e->cap));
        if (mincap) {
            g[e->v][e->rev].cap += mincap;
            e->cap -= mincap;
            fu -= mincap;
        }
    }
    if (f == fu)
        level[u] = -1;
    return f - fu;
}
T max_flow(int src, int sink) {
    flow = 0;
    while (build_level_graph(src, sink))
        flow += blocking_flow(src, sink,
            numeric_limits<T>::max());
    return flow;
}
};
```

2.13 Dinic Bruno

```
// tem que definir o tamanho de g e de lev como o numero
// de vertices do grafo e depois char o a funcao fluxo
//
// Complexidade:
// Caso geral: O(V^2 * E)
// Grafo bipartido O(sqrt(V)*E)

#define INF 0x3f3f3f3f

struct edge{
    int p, c, id; // destino, capacidade, id
    edge() {p = c = id = 0;}
    edge(int p, int c, int id):p(p), c(c), id(id){}
};

vector<vector<edge> > g; // define o tamanho depois
vector <int> lev;

void add(int a, int b, int c){
    // de a para b com capacidade c
    edge d = {b, c, (int) g[b].size()};
    edge e = {a, 0, (int) g[a].size()};
    g[a].pb(d);
    g[b].pb(e);
}

bool bfs(int s, int t){
    // bfs de s para t construindo o level
    for(int i = 0; i < g.size(); i++)
        lev[i] = -1;
    lev[s] = 0;

    // bfs saindo de s
    queue <int> q;
    q.push(s);
    while(q.size()){
        int u = q.front(); q.pop();

        for(int i = 0; i < g[u].size(); i++){
            edge e = g[u][i];
            // se ja foi visitado ou nao tem capacidade nao
                visita
            if(lev[e.p] != -1 || !e.c) continue;
            lev[e.p] = lev[u] + 1;
            if(e.p == t) return true;
            q.push(e.p);
        }
```

```
    }

    return false;
}

int dfs(int v, int s, int f){
    if(v == s || !f) return f;

    int flu = f;
    for(int i = 0; i < g[v].size(); i++){
        edge e = g[v][i]; int u = e.p;

        // visita se tiver capacidadade e se ta no proximo
           nivel
        if(lev[u] != lev[v] + 1 || !e.c) continue;

        int tenta = dfs(u, s, min(flu, e.c));
        // se passou alguma coisa altera as capacidades
        if(tenta){
            flu -= tenta;
            g[v][i].c -= tenta;
            g[u][e.id].c += tenta;
        }
    }

    // se passou tudo tira da lista dos possiveis
    if(flu == f) lev[v] = -1;
    return f - flu;
}

int fluxo(int s, int t){
    int r = 0;
    while(bfs(s, t)) r += dfs(s, t, INF);
    return r;
}

// ja tem ate o debug
void imprime(){
    for(int i = 0; i < g.size(); i++){
        printf("%i -> ", i);
        for(int j = 0; j < g[i].size(); j++)
            printf("(%i %i)", g[i][j].p, g[i][j].c);
```

```
            printf("\n");
        }
        printf("\n");
}
```

2.14 Kosaraju

```
// O(n + m)

int n;
vector<vector<int> > g(MAX);
vector<vector<int> > gi(MAX); // grafo invertido
int vis[MAX];
stack<int> S;
int comp[MAX]; // componente conexo de cada vertice

void dfs(int k) {
    vis[k] = 1;
    for (int i = 0; i < (int) g[k].size(); i++)
        if (!vis[g[k][i]]) dfs(g[k][i]);

    S.push(k);
}

void scc(int k, int c) {
    vis[k] = 1;
    comp[k] = c;
    for (int i = 0; i < (int) gi[k].size(); i++)
        if (!vis[gi[k][i]]) scc(gi[k][i], c);
}

void kosaraju() {
    for (int i = 0; i < n; i++) vis[i] = 0;
    for (int i = 0; i < n; i++) if (!vis[i]) dfs(i);

    for (int i = 0; i < n; i++) vis[i] = 0;
    while (S.size()) {
        int u = S.top();
        S.pop();
        if (!vis[u]) scc(u, u);
    }
}
```

2.15 Kruskal

```
// Gera AGM a partir do vetor de arestas
//
// O(m log(n))

int n;
vector<pair<int, pair<int, int> > > ar; // vetor de arestas
int v[MAX];

// Union-Find em O(log(n))
void build();
int find(int k);
void une(int a, int b);

void kruskal() {
    build();

    sort(ar.begin(), ar.end());
    for (int i = 0; i < (int) ar.size(); i++) {
        int a = ar[i].s.f, b = ar[i].s.s;
        if (find(a) != find(b)) {
            une(a, b);
            // aresta faz parte da AGM
        }
    }
}
```

2.16 Ponte

```
// Chama zera(numDeVertices)
// Depois dfs para (0, -1) = (verticeInicial, paiDele)
// Se tiver ponte a variavel ok vai ser 0 no final
//
// Complexidade: O(n + m)

vector <vector<int> > g(N);
vector<int> di (N); // distancia do vertice inicial
vector<int> lo (N); // di do menor vertice que ele alcanca
vector<int> vi (N);
int d, ok;

void zera(int n){
    for(int i = 0; i < n; i++){
        g[i].clear();
        di[i] = -1;
        lo[i] = INF;
        vi[i] = 0;
    }
    ok = 1;
    d = 0;
}

void dfs(int v, int pai){
    vi[v] = 1;
    // ele eh o d-esimo a ser visitado e alcanca o d-esimo
       vertice
    di[v] = lo[v] = d++;

    for(int i = 0; i < g[v].size(); i++){
        int u = g[v][i];
        if(!vi[u]) dfs(u, v);

        // o filho nao alcanca ninguem menor ou igual a ele,
           eh ponte
        if(di[v] < lo[u]) ok = 0;

        // atualiza o menor que ele alcanca
        if(pai != u && lo[u] < lo[v])
            lo[v] = lo[u];
    }
}
```

2.17 Tarjan

```
// O(n + m)

int n;
vector<vector<int> > g(MAX);
stack<int> s;
int vis[MAX], comp[MAX];
int id[MAX], p;

int dfs(int k) {
```

```
    int lo = id[k] = p++;
    s.push(k);
    vis[k] = 2; // ta na pilha

    // calcula o menor cara q ele alcanca
    // que ainda nao esta em um scc
    for (int i = 0; i < g[k].size(); i++) {
        if (!vis[g[k][i]])
            lo = min(lo, dfs(g[k][i]));
        else if (vis[g[k][i]] == 2)
            lo = min(lo, id[g[k][i]]);
    }

    // nao alcanca ninguem menor -> comeca scc
    if (lo == id[k]) while (1) {
        int u = s.top();
        s.pop(); vis[u] = 1;
        comp[u] = k;
        if (u == k) break;
    }

    return lo;
}

void tarjan() {
    memset(vis, 0, sizeof(vis));

    p = 0;
    for (int i = 0; i < n; i++) if (!vis[i]) dfs(i);
}
```

3 Matemática

3.1 Miller-Rabin

```
// Testa se n eh primo, n <= 3 * 10^18
//
// O(log(n)), considerando multiplicacao
// e exponenciacao constantes

// multiplicacao modular
ll mul(ll x, ll y, ll m); // x*y mod m
ll exp(ll x, ll y, ll m); // x^y mod m;

bool prime(ll n) {
    if (n < 2) return 0;
    if (n <= 3) return 1;
    if (n % 2 == 0) return 0;

    ll d = n - 1;
    int r = 0;
    while (d % 2 == 0) r++, d /= 2;

    // com esses primos, o teste funciona garantido para n
        <= 3*10^18
    // funciona para n <= 3*10^24 com os primos ate 41
    int a[9] = {2, 3, 5, 7, 11, 13, 17, 19, 23};
    // outra opcao para n <= 2^64:
    // int a[7] = {2, 325, 9375, 28178, 450775, 9780504,
        1795265022};

    for (int i = 0; i < 9; i++) {
        if (a[i] >= n) break;
        ll x = exp(a[i], d, n);
        if (x == 1 or x == n - 1) continue;

        bool deu = 1;
        for (int j = 0; j < r - 1; j++) {
            x = mul(x, x, n);
            if (x == n - 1) {
                deu = 0;
                break;
            }
        }
        if (deu) return 0;
    }
    return 1;
}
```

3.2 Crivo de Erastosthenes

```
// "O" crivo
```

```
//
// Encontra maior divisor primo
// Um numero eh primo sse div[x] == x
// fact fatora um numero <= lim
// A fatoracao sai ordenada
//
// crivo - O(n log(log(n)))
// fact - O(log(n))

int divi[MAX];

void crivo(int lim) {
    for (int i = 1; i <= lim; i++) divi[i] = 1;

    for (int i = 2; i <= lim; i++) if (divi[i] == 1)
        for (int j = i; j <= lim; j += i) divi[j] = i;
}

void fact(vector<int>& v, int n) {
    if (n != divi[n]) fact(v, n/divi[n]);
    v.push_back(divi[n]);
}

// Crivo de divisores
//
// Encontra numero de divisores
// ou soma dos divisores
//
// O(n log(n))

int divi[MAX];

void crivo(int lim) {
    for (int i = 1; i <= lim; i++) divi[i] = 1;

    for (int i = 2; i <= lim; i++)
        for (int j = i; j <= lim; j += i) {
            // para numero de divisores
            divi[j]++;
            // para soma dos divisores
            divi[j] += i;
        }
}

// Crivo de totiente
//
// Encontra o valor da funcao
// totiente de Euler
//
// O(n log(log(n)))

int tot[MAX];

void crivo(int lim) {
    for (int i = 1; i <= lim; i++) tot[i] = i;

    for (int i = 2; i <= lim; i++) if (tot[i] == i)
        for (int j = i; j <= lim; j += i)
            tot[j] -= tot[j] / i;
}
```

3.3 Exponenciação rápida

```
// (x^y mod m) em O(log(y))

typedef long long int ll;

ll pow(ll x, ll y, ll m) { // iterativo
    ll ret = 1;
    while (y) {
        if (y & 1) ret = (ret * x) % m;
        y >>= 1;
        x = (x * x) % m;
    }
    return ret;
}

ll pow(ll x, ll y, ll m) { // recursivo
    if (y == 0) return 1;

    ll ret = pow(x, y / 2, m);
    ret = (ret * ret) % m;
    if (y & 1) ret = (ret * x) % m;
    return ret;
```

```
}
```

3.4 Euclides

```
// O(log(min(a, b)))

int mdc(int a, int b) {
    return !b ? a : mdc(b, a % b);
}
```

3.5 Euclides extendido

```
// acha x e y tal que ax + by = mdc(a, b)
//
// O(log(min(a, b)))

int mdce(int a, int b, int *x, int *y){
    if(!a){
        *x = 0;
        *y = 1;
        return b;
    }

    int X, Y;
    int mdc = mdce(b % a, a, &X, &Y);
    *x = Y - (b / a) * X;
    *y = X;

    return mdc;
}
```

3.6 Inverso Modular

```
// Computa o inverso de a modulo b
// Se b eh primo, basta fazer
// a^(b-2)

long long inv(long long a, long long b){
    return 1<a ? b - inv(b%a,a)*b/a : 1;
}
```

3.7 Ordem Grupo

```
// O grupo Zn eh ciclico sse n =
// 1, 2, 4, p^k ou 2 p^k, p primo impar
// Retorna -1 se nao achar
//
// O(sqrt(n) log(n))

int tot(int n); // totiente em O(sqrt(n))
int expo(int a, int b, int m); // (a^b)%m em O(log(b))

// acha todos os divisores ordenados em O(sqrt(n))
vector<int> div(int n) {
    vector<int> ret1, ret2;
    for (int i = 1; i*i <= n; i++) if (n % i == 0) {
        ret1.pb(i);
        if (i*i != n) ret2.pb(n/i);
    }

    for (int i = ret2.size()-1; i+1; i--) ret1.pb(ret2[i]);
    return ret1;
}

int ordem(int a, int n) {
    vector<int> v = div(tot(n));
    for (int i : v) if (expo(a, i, n) == 1) return i;
    return -1;
}
```

3.8 Pollard's Rho

```
// Usa o algoritmo de deteccao de ciclo de Brent
// A fatoracao nao sai necessariamente ordenada
// O algoritmo rho encontra um fator de n,
// e funciona muito bem quando n possui um fator pequeno
// Eh recomendado chamar srand(time(NULL)) na main
//
// Complexidades (considerando mul constante):
// rho - esperado O(n^(1/4)) no pior caso
// fact - esperado menos que O(n^(1/4) log(n)) no pior caso

ll mdc(ll a, ll b) { return !b ? a : mdc(b, a % b); }
```

```
ll mul(ll x, ll y, ll m) {
    if (!y) return 0;

    ll ret = mul(x, y >> 1, m);
    ret = (ret + ret) % m;
    if (y & 1) ret = (ret + x) % m;
    return ret;
}

ll exp(ll x, ll y, ll m) {
    if (!y) return 1;

    ll ret = exp(x, y >> 1, m);
    ret = mul(ret, ret, m);
    if (y & 1) ret = mul(ret, x, m);
    return ret;
}

bool prime(ll n) {
    if (n < 2) return 0;
    if (n <= 3) return 1;
    if (n % 2 == 0) return 0;

    ll d = n - 1;
    int r = 0;
    while (d % 2 == 0) {
        r++;
        d /= 2;
    }

    int a[9] = {2, 3, 5, 7, 11, 13, 17, 19, 23};
    for (int i = 0; i < 9; i++) {
        if (a[i] >= n) break;
        ll x = exp(a[i], d, n);
        if (x == 1 or x == n - 1) continue;

        bool deu = 1;
        for (int j = 0; j < r - 1; j++) {
            x = mul(x, x, n);
            if (x == n - 1) {
                deu = 0;
                break;
            }
        }
        if (deu) return 0;
    }
    return 1;
}

ll rho(ll n) {
    if (n == 1 or prime(n)) return n;
    if (n % 2 == 0) return 2;

    while (1) {
        ll x = 2, y = 2;
        ll ciclo = 2, i = 0;

        ll c = (rand() / (double) RAND_MAX) * (n - 1) + 1;
        ll d = 1;

        while (d == 1) {
            if (++i == ciclo) ciclo *= 2, y = x;
            x = (mul(x, x, n) + c) % n;

            if (x == y) break;

            d = mdc(abs(x - y), n);
        }

        if (x != y) return d;
    }
}

void fact(ll n, vector<ll>& v) {
    if (n == 1) return;
    if (prime(n)) v.pb(n);
    else {
        ll d = rho(n);
        fact(d, v);
        fact(n / d, v);
    }
}
```

3.9 Totiente

```
// O(sqrt(n))

int tot(int n){
    int ret = n;

    for (int i = 2; i*i <= n; i++) if (n % i == 0) {
        while (n % i == 0) n /= i;
        ret -= ret / i;
    }
    if (n > 1) ret -= ret / n;

    return ret;
}
```

4 Problemas

4.1 Area Histograma

```
// Assume que todas as barras tem largura 1,
// e altura dada no vetor v
//
// O(n)

typedef long long ll;

ll area(vector<int> v) {
    ll ret = 0;
    stack<int> s;
    // valores iniciais pra dar tudo certo
    v.insert(v.begin(), -1);
    v.insert(v.end(), -1);
    s.push(0);

    for(int i = 0; i < (int) v.size(); i++) {
        while (v[s.top()] > v[i]) {
            ll h = v[s.top()]; s.pop();
            ret = max(ret, h * (i - s.top() - 1));
        }
        s.push(i);
    }

    return ret;
}
```

4.2 Convex Hull Trick

```
// linear

struct CHT {
    int it;
    vector<ll> a, b;
    CHT():it(0){}
    ll eval(int i, ll x){
        return a[i]*x + b[i];
    }
    bool useless(){
        int sz = a.size();
        int r = sz-1, m = sz-2, l = sz-3;
        return  (b[l] - b[r])*(a[m] - a[l]) <
            (b[l] - b[m])*(a[r] - a[l]);
    }
    void add(ll A, ll B){
        a.push_back(A); b.push_back(B);
        while (!a.empty()){
            if ((a.size() < 3) || !useless()) break;
            a.erase(a.end() - 2);
            b.erase(b.end() - 2);
        }
    }
    ll get(ll x){
        it = min(it, int(a.size()) - 1);
        while (it+1 < a.size()){
            if (eval(it+1, x) > eval(it, x)) it++;
            else break;
        }
        return eval(it, x);
    }
};
```

4.3 Inversion Count

```
// O(n log(n))

int n;
int v[MAX];

// bit de soma
void poe(int p);
int query(int p);

// converte valores do array pra
// numeros de 1 a n
void conv() {
    vector<int> a;
    for (int i = 0; i < n; i++) a.push_back(v[i]);

    sort(a.begin(), a.end());

    for (int i = 0; i < n; i++)
        v[i] = 1 + (lower_bound(a.begin(), a.end(), v[i]) -
            a.begin());
}

long long inv() {
    conv();
    build();

    long long ret = 0;
    for (int i = n - 1; i >= 0; i--) {
        ret += query(v[i] - 1);
        poe(v[i]);
    }
    return ret;
}
```

4.4 LIS 1

```
// Calcula uma LIS
// Para ter o tamanho basta fazer lis().size()
// Implementacao do algotitmo descrito em:
// https://goo.gl/HiFkn2
//
// O(n log(n))

const int INF = 0x3f3f3f3f;

int n, v[MAX];

vector<int> lis() {
    int I[n + 1], L[n];

    // pra BB funfar bacana
    I[0] = -INF;
    for (int i = 1; i <= n; i++) I[i] = INF;

    for (int i = 0; i < n; i++) {
        // BB
        int l = 0, r = n;
        while (l < r) {
            int m = (l + r) / 2;
            if (I[m] >= v[i]) r = m;
            else l = m + 1;
        }

        // ultimo elemento com tamanho l eh v[i]
        I[l] = v[i];
        // tamanho da LIS terminando com o
        // elemento v[i] eh l
        L[i] = l;
    }

    // reconstroi LIS
    vector<int> ret;
    int m = -INF, p;
    for (int i = 0; i < n; i++) if (L[i] > m) {
        m = L[i];
        p = i;
    }
    ret.push_back(v[p]);
    int last = m;
    while (p--) if (L[p] == m - 1) {
        ret.push_back(v[p]);
        m = L[p];
```

```
    }

    reverse(ret.begin(), ret.end());
    return ret;
}
```

4.5 LIS 2

```
// O(n log(n))

template<typename T> int lis(vector<T> &v){
    vector<T> ans;
    for (T t : v){
        auto it = upper_bound(ans.begin(), ans.end(), t);
        if (it == ans.end()) ans.push_back(t);
        else *it = t;
    }
    return ans.size()
}
```

4.6 Merge Sort

```
// Melhor do Brasil, segundo o autor
//
// O(n log(n))

long long merge_sort(int l, int r, vector<int> &t){
    if (l >= r) return 0;
    int m = (l+r)/2;
    auto ans = merge_sort(l, m, t) + merge_sort(m+1, r, t);
    static vector<int> aux; if (aux.size() != t.size())
       aux.resize(t.size());
    for (int i = l; i <= r; i++) aux[i] = t[i];

    int i_l = l, i_r = m+1, i = l;
    auto move_l = [&](){
        t[i++] = aux[i_l++];
    };
    auto move_r = [&](){
        t[i++] = aux[i_r++];
    };
    while (i <= r){
        if (i_l > m) move_r();
        else if (i_r > r) move_l();
        else{
            if (aux[i_l] <= aux[i_r]) move_l();
            else{
                move_r();
                ans += m - i_l + 1;
            }

        }
    }
    return ans;
}
```

4.7 MO

```
// O(n sqrt(n) + q)

void add(int pos){
    occ[a[pos]]++;
    counter += (occ[a[pos]] == 1);
}

void remove(int pos){
    occ[a[pos]]--;
    counter -= (occ[a[pos]] == 0);
}

sort(s.begin(), s.end()); //sort queries
for (int i = 0; i < q; i++){
    int iq = s[i].second;
    pii q = query[iq];
    while (L < q.first){
        remove(L);
        L++;
    }
    while (L > q.first){
        L--;
        add(L);
    }
```

```
    while (R < q.second){
        R++;
        add(R);
    }
    while (R > q.second){
        remove(R);
        R--;
    }
    ans[iq] = counter;
}
```

4.8 Nim

```
// Calcula movimento otimo do jogo classico de Nim
// Assume que o estado atual eh perdedor
// Funcao move retorna um par com a pilha (0 indexed)
// e quanto deve ser tirado dela
// XOR deve estar armazenado em x
// Para mudar um valor, faca insere(novo_valor),
// atualize o XOR e mude o valor em v
//
// MAX2 = teto do log do maior elemento
// possivel nas pilhas
//
// O(log(n)) amortizado

int v[MAX], n, x;
stack<int> pi[MAX2];

void insere(int p) {
    for (int i = 0; i < MAX2; i++) if (v[p] & (1 << i))
        pi[i].push(p);
}

pair<int, int> move() {
    int bit = 0; while (x >> bit) bit++; bit--;

    // tira os caras invalidos
    while ((v[pi[bit].top()] & (1 << bit)) == 0)
        pi[bit].pop();

    int cara = pi[bit].top();
    int tirei = v[cara] - (x^v[cara]);
    v[cara] -= tirei;

    insere(cara);

    return make_pair(cara, tirei);
}

// Acha o movimento otimo baseado
// em v apenas
//
// O(n)

pair<int, int> move() {
    int x = 0;
    for (int i = 0; i < n; i++) x ^= v[i];

    for (int i = 0; i < n; i++) if ((v[i]^x) < v[i])
        return make_pair(i, v[i] - (v[i]^x));
}
```

5 String

5.1 Hashing

```
// String hashing
//
// String deve ter valores [1, x]
// p deve ser o menor primo maior que x
// Para evitar colisao: testar mais de um
// mod; so comparar strings do mesmo tamanho
// ex : str_hash<31, 1e9+7> h(s);
//      ll val = h(10, 20);
//
// Complexidades:
// build - O(|s|)
// get_hash - O(1)

typedef long long ll;
```

```cpp
template<int P, int MOD> struct str_hash {
    int n;
    string s;
    vector<ll> h, power;
    str_hash(string s_): n(s_.size()), s(s_), h(n), power(n){
        power[0] = 1;
        for (int i = 1; i < n; i++) power[i] = power[i-1]*p
            % m;
        h[0] = s[0];
        for (int i = 1; i < n; i++) h[i] = (h[i-1]*p + s[i])
            % m;
    }
    ll operator()(int i, int j){
        if (!i) return h[j];
        return (h[j] - h[i-1]*power[j-i+1] % m + m) % m;
    }
};
```

5.2 KMP

```cpp
// Primeiro chama a funcao process com o padrao
// Depois chama match com (texto, padrao)
// Vai retornar o numero de ocorrencias do padrao
//
// Complexidades:
// process - O(m)
// match - O(n + m)
// n = |texto| e m = |padrao|

int p[N];

void process(string &s){
    int i = 0, j = -1;
    p[0] = -1;
    while(i < s.size()){
        while(j >= 0 and s[i] != s[j]) j = p[j];
        i++; j++;
        p[i] = j;
    }
}

int match(string &s, string &t){
    int r = 0;
    process(t);
    int i = 0, j = 0;
    while(i < s.size()){
        while(j >= 0 and s[i] != t[j]) j = p[j];
        i++; j++;
        if(j == t.size()){
            j = p[j];
            r++;
        }
    }
    return r;
}
```

5.3 SuffixArray 1

```cpp
// Suffix Array
//
// kasai recebe o suffix array e calcula lcp[i],
// o lcp entre s[sa[i],...,n-1] e s[sa[i+1],..,n-1]
//
// Complexidades:
// suffix_array - O(n log(n))
// kasai - O(n)

vector<int> suffix_array(string s) {
    s += "$";
    int n = s.size(), N = max(n, 260);
    vector<int> sa(n), ra(n);
    for(int i = 0; i < n; i++) sa[i] = i, ra[i] = s[i];

    for(int k = 0; k < n; k ? k *= 2 : k++) {
        vector<int> nsa(sa), nra(n), cnt(N);

        for(int i = 0; i < n; i++) nsa[i] = (nsa[i]-k+n)%n,
            cnt[ra[i]]++;
        for(int i = 1; i < N; i++) cnt[i] += cnt[i-1];
        for(int i = n-1; i+1; i--) sa[--cnt[ra[nsa[i]]]] =
            nsa[i];

        for(int i = 1, r = 0; i < n; i++) nra[sa[i]] = r +=
            ra[sa[i]] !=
```

```
            ra[sa[i-1]] or ra[(sa[i]+k)%n] !=
               ra[(sa[i-1]+k)%n];
        ra = nra;
    }
    return vector<int>(sa.begin()+1, sa.end());
}

vector<int> kasai(string s, vector<int> sa) {
    int n = s.size(), k = 0;
    vector<int> ra(n), lcp(n);
    for (int i = 0; i < n; i++) ra[sa[i]] = i;

    for (int i = 0; i < n; i++, k -= !!k) {
        if (ra[i] == n-1) { k = 0; continue; }
        int j = sa[ra[i]+1];
        while (i+k < n and j+k < n and s[i+k] == s[j+k]) k++;
        lcp[ra[i]] = k;
    }
    return lcp;
}
```

5.4 SuffixArray 2

```
// Suffix Array Rafael
//
// O(n log^2(n))

struct suffix_array{
    string &s;
    int n;
    vector<int> p, r, aux, lcp;
    seg_tree<int, min_el> st;
    suffix_array(string &s):
        s(s), n(s.size()), p(n), r(n), aux(n), lcp(n){
            for (int i = 0; i < n; i++){
                p[i] = i;
                r[i] = s[i];
            }
            auto rank = [&](int i){
                if (i >= n) return -i;
                return r[i];
            };
            for (int d = 1; d < n; d *= 2){
                auto t = [&](int i){
                    return make_pair(rank(i), rank(i+d));
                };
                sort(p.begin(), p.end(),
                        [&](int &i, int &j){
                        return t(i) < t(j);
                        }
                    );
                aux[p[0]] = 0;
                for (int i = 1; i < n; i++)
                    aux[p[i]] = aux[p[i-1]] + (t(p[i]) >
                       t(p[i-1]));
                for (int j = 0; j < n; j++) r[j] = aux[j];
                if (aux[p[n-1]] == n-1) break;
            }

            int h = 0;
            for (int i = 0; i < n; i++){
                if (r[i] == n-1){
                    lcp[r[i]] = 0;
                    continue;
                }
                int j = p[r[i] + 1];
                while (i + h < n && j + h < n && s[i+h] ==
                   s[j+h]) h++;
                lcp[r[i]] = h;
                h = max(0, h-1);
            }
            st = seg_tree<int, min_el>(&lcp);
        }
    int query(int l, int r){
        return st.query(l, r);
    }
    ll distinct_substrings(){
        ll ans = p[0] + 1;
        for (int i = 1; i < n; i++)
            ans += p[i] - lcp[i-1] + 1;
        return ans;
    }
};
```

5.5 Z

```
// Complexidades:
// z - O(|s|)
// match - O(|s| + |p|)

vector<int> get_z(string s) {
    int n = s.size();
    vector<int> z(n, 0);

    // intervalo da ultima substring valida
    int l = 0, r = 0;
    for (int i = 1; i < n; i++) {
        // estimativa pra z[i]
        if (i <= r) z[i] = min(r - i + 1, z[i - l]);
        // calcula valor correto
        while (i + z[i] < n and s[z[i]] == s[i + z[i]])
            z[i]++;
        // atualiza [l, r]
        if (i + z[i] - 1 > r) l = i, r = i + z[i] - 1;
    }

    return z;
}

// quantas vezes p aparece em s
int match(string s, string p) {
    int n = s.size(), m = p.size();
    vector<int> z = get_z(p + s);

    int ret = 0;
    for (int i = m; i < n + m; i++)
        if (z[i] >= m) ret++;

    return ret;
}
```

6 Extra

6.1 vimrc

```
set ts=4 si ai sw=4 number mouse=a
syntax on
```

6.2 Makefile

```
CXX = g++
CXXFLAGS = -O2 -Wall -Wshadow -std=c++11 -Wno-unused-result
   -Wno-sign-compare
```

6.3 Template

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

using namespace std;

#define _ ios_base::sync_with_stdio(0);cin.tie(0);
#define endl '\n'
#define sc(a) scanf("%d",&a)
#define sc2(a,b) sc(a), sc(b)
#define pri(x) printf("%d\n",x)
#define f first
#define s second
#define pb push_back

typedef long long ll;
typedef pair<int, int> ii;

const int INF = 0x3f3f3f3f;
const ll LINF = 0x3f3f3f3f3f3f3f3f;

int main(){ _
    exit(0);
}
```