Competitive Programming Notebook

Miguel Nogueira

Contents

1 Dynamic programming

1.1 Optimal-selection

```cpp
/*
    Optimal Selection
    Tens n escolhas pra fazer em k intervalos de tempo,
    escolhe o melhor subconjunto tal que alguma heuristica
    eh maximizada ao longo de todos os timestamps
*/

int optimal_selection(int n, int k, int w[][]) {
   int f[(1 << n)][n + 1];
    for (int i = 0; i < (1 << n); i++) {
        for (int j = 0; j < n; j++) {
            f[i][j] = 0;
        }
    }
    for (int i = 0; i < k; i++) {
        f[(1 << i)][0] = w[i][0];
    }
    for (int day = 1; day < n; day++) {
        for (int mask = 0; mask < 8; mask++) {
            f[mask][day] = f[mask][day - 1];
            for (int y = 0; y < k; y++) {
                if (mask & (1 << y)) {
                    f[mask][day] = max(f[mask][day], f[mask^(1 << k)][day - 1] + w[k][day]);
                }
            }
        }
    }
    return f[(1 << n) - 1][n - 1];
}
```

1.2 Longest-increasing-subsequence

```cpp
/*
    Longest Increasing Subsequence
    Encontra o tamanho e recupera uma LIS de um vetor `a`
    Complexidade: O(n log n)
*/

vector<int> lis(vector<int> const& a) {
    int n = a.size();
    vector<int> d(n+1, INF), pos(n+1, -1), prev(n, -1);
    d[0] = -INF;

    for (int i = 0; i < n; i++) {
        int l = lower_bound(d.begin(), d.end(), a[i]) - d.begin();
        if (a[i] < d[l]) {
            d[l] = a[i];
            pos[l] = i;
            prev[i] = pos[l-1];
        }
    }

    int len = 0;
    while (d[len] < INF) {
        len++;
    }
    len--;

    vector<int> result;
    int curr_pos = pos[len];
    while (curr_pos != -1) {
        result.push_back(a[curr_pos]);
        curr_pos = prev[curr_pos];
    }
    reverse(result.begin(), result.end());

    return result;
}
```

1.3 Digit

```cpp
#include <bits/stdc++.h>
#define ll long long
using namespace std;

/*
Digit DP
Calcula a soma dos digitos de todos os numeros entre 0 e `number`
para intervalo [a, b] -> solve(b) - solve(a - 1)
*/

const int MAX_DIGITS = 10;
ll dp[MAX_DIGITS][180][2];
vector<int> number;
ll f(int pos, ll sum, int smaller) {
    if (pos == number.size()) return sum;
    ll &ans = dp[pos][sum][smaller];
    if (~ans) return ans;
    ans = 0;
    for (int i=0; i <= (smaller ? 9: number[pos]); i++) {
        bool smaller_now = (smaller || i < number[pos]);
        ans += f(pos + 1, sum + i, smaller_now);
    }
    return dp[pos][sum][smaller] = ans;
}

/*

    Se nao tiver inversa:
    const int MAX_DIGITS = 20;
    const int MAX_K = 20;
    ll dp[MAX_DIGITS][MAX_K][2][2]; //

    int d, k;
    vector<int> number_a, number_b;

    ll solve(int pos, int cnt, bool smaller_than_b, bool greater_than_a){
        if(pos == number_a.size()) return (cnt == k);
        ll &ans = dp[pos][cnt][smaller_than_b][greater_than_a];
        if(~ans) return ans;
        ans = 0;
        for(int i = (greater_than_a ? 0 : number_a[pos]); i <= (smaller_than_b ? 9 : number_b[pos]); i++){
            bool is_smaller_now = (smaller_than_b || (i < number_b[pos]));
            bool is_greater_now = (greater_than_a || (i > number_a[pos]));
            int new_cnt = cnt + (i == d);
            ans += solve(pos+1, new_cnt, is_smaller_now, is_greater_now);
        }
        return ans;
    }

*/

vector<int> ntovec(int num)  {
    if (num == 0) return {0};
    vector<int> v;
```

```cpp
    for (; num >0; num /= 10) v.push_back(num % 10);
    reverse(begin(v), end(v));
    return v;
}

ll solve(int n) {
    if (n < 0) return 0;
    number = ntovec(n);
    memset(dp, -1, sizeof dp);
    return f(0, 0, false);
}

ll ans(int a, int b) {
    return solve(b) - solve(a - 1);
}
```

1.4 Knapsack

```cpp
#include <bits/stdc++.h>
#define ll long long
using namespace std;

/*
    Knapsack Problem -
    Given a set of items with value i and cost j, and you have limited budget
    find the subset of items you can take where total value is maximal
    Variations covered:
        - 0/1 Knapsack - Only one copie of each item can be taken
        - Bounded Knapsack - Each item has a number k[i] of copies

    If item retrieval is unecessary prefer 1D knapsack
*/

/*
    0/1 Knapsack - One copy of each item
*/

int f[n + 1][cap + 1], weight[n], value[n];

// Time: O(nW)
// Space : O(nW)
int knapsack_2D() {
    for (int i = 1; i <= n; i++) {
        for (int w = 0; w <= W; w++) {
            f[i][w] = f[i - 1][w];
            if (w >= weight[i - 1]) {
                f[i][w] = max(f[i][w], f[i - 1][w - weight[i - 1]] + value[i - 1]);
            }
        }
    }
    return f[n][W];
}

// Time O(nW)
// Space O(w)
int knapsack_1D() {
    for (int i = 0; i < n; i++) {
        for (int w = W; w >= weight[i]; i--) {
            f[w] = max(f[w], f[w - weight[i]] + value[i]);
        }
    }
    return f[W];
}

/*
Unbounded Knapsack - Infinite copies of each item
*/

// Time O(nW)
// Space O(nw)
int unbounded_knapsack_2D() {
    for (int i = 1; i <= n; i++) {
        for (int w = 0; w <= W; w++) {
            f[i][w] = f[i - 1][w];  // Not taking item
            if (w >= weight[i - 1]) {
                f[i][w] = max(f[i][w], f[i][w - weight[i - 1]] + value[i - 1]);
            }
        }
    }
    return f[n][W];
}

// Time O(nW)
// Space O(W)
int unbounded_knapsack_1D() {
    for (int i = 0; i < n; i++) {
        for (int w = weight[i]; w <= W; w++) {  // Forward loop allows reuse
            f[w] = max(f[w], f[w - weight[i]] + value[i]);
        }
    }
    return f[W];
}

/*
    Bounded knapsack - Bounded number of copies of each item
*/

// Time: O(nWk) suitable for small k
// Space: O(nW)
int bounded_knapsack_2D() {
    for (int i = 1; i <= n; i++) {
        for (int w = 0; w <= W; w++) {
            f[i][w] = f[i - 1][w];  // Not taking item
            for (int k = 1; k <= count[i - 1] && k * weight[i - 1] <= w; k++) {
                f[i][w] = max(f[i][w], f[i - 1][w - k * weight[i - 1]] + k * value[i - 1]);
            }
        }
    }
    return f[n][W];
}

// Time: O(nW) any k
// Space: O(W)
int bounded_knapsack_1D() {
    for (int i = 0; i < n; i++) {
        for (int k = 1; count[i] > 0; k *= 2) {
            int take = min(k, count[i]);
            count[i] -= take;
            for (int w = W; w >= take * weight[i]; w--) {
                f[w] = max(f[w], f[w - take * weight[i]] + take * value[i]);
            }
        }
    }
    return f[W];
}
```

1.5 Sos

```cpp
// F[mask] = sum of values of all submasks of mask
```

```
for (int i = 0; i < n; i++) {
    for (int mask = 0; mask < (1 << n); mask++) {
        if (mask & (1 << i)) {
            dp[mask] += dp[mask ^ (1 << i)];
        }
    }
}
```

2 General

2.1 Progressions

```
ll nthTermAP(ll a, ll d, int n) {
    return a + (n - 1) * d;
}

ll sumAP(ll a, ll d, int n) {
    return (n / 2LL) * (2LL * a + (n - 1) * d);
}

ll nthTermGP(ll a, ll r, int n) {
    return a * pow(r, n - 1);
}

ll sumGP(ll a, ll r, int n) {
    if (r == 1) return a * n; // Special case for r=1
    return a * (1 - pow(r, n)) / (1 - r);
}
```

2.2 Gray Code

```
// Generate gray code sequence for n bits
for (int i = 0; i < (1 << n); i++) {
    int gray = i ^ (i >> 1);
    // Process gray code
}
```

2.3 Rng

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

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

3 Geometry

3.1 Convex-hull

```
struct Point {
    ll x, y;
    Point(ll x=0, ll y=0) : x(x), y(y) {}
    Point operator+ (const Point&a) const{ return Point(x+a.x, y+a.y); }
    Point operator- (const Point&a) const{ return Point(x-a.x, y-a.y); }
    ll operator* (const Point&a) const{ return (x*a.x + y*a.y); } //DOT product // norm // lenght^2 // inner
    ll operator% (const Point&a) const{ return (x*a.y - y*a.x); } //Cross // Vector product
    Point operator* (ll c) const{ return Point(x*c, y*c); }
    Point operator/ (ll c) const{ return Point(x/c, y/c); }

    bool operator==(const Point&a) const{ return x == a.x && y == a.y; }
    bool operator< (const Point&a) const{ return x != a.x ? x < a.x : y < a.y; }
    bool operator<<(const Point&a) const{ Point p=*this; return (p%a == 0) ? (p*p < a*a) : (p%a < 0); } //angle(p) < angle(a)
}

/*******************
// FOR DOUBLE POINT //
const ld EPS = 1e-9;
bool eq(ld a, ld b){ return abs(a-b) < EPS; } // ==
bool lt(ld a, ld b){ return a + EPS < b;    } // <
bool gt(ld a, ld b){ return a > b + EPS;    } // >
bool le(ld a, ld b){ return a < b + EPS;    } // <=
bool ge(ld a, ld b){ return a + EPS > b;    } // >=
bool operator==(const PT&a) const{ return eq(x, a.x) && eq(y, a.y); }                // for double point
bool operator< (const PT&a) const{ return eq(x, a.x) ? lt(y, a.y) : lt(x, a.x); }   // for double point
bool operator<<(PT&a){ PT&p=*this; return eq(p%a, 0) ? lt(p*p, a*a) : lt(p%a, 0);  } //angle(this) < angle(a)
//Change LL to LD and uncomment this
//Also, consider replacing comparisons with these functions
********************/

vector<Point> ch(vector<Point> pts, bool sorted=false) {
    if(!sorted) sort(begin(pts), end(pts));
    pts.resize(unique(begin(pts), end(pts)) - begin(pts));
    if(pts.size() <= 1) return pts;
    int s = 0, n = pts.size();
    vector<Point> h (2 * n + 1);
    for(int i=0; i<n; h[s++] = pts[i++])
        while(s > 1 && (pts[i] - h[s-2]) % (h[s-1] - h[s-2]) > 0 )
            s--;
    for(int i=n-2, t=s; ~i; h[s++] = pts[i--])
        while(s > t && (pts[i] - h[s-2]) % (h[s-1] - h[s-2]) > 0 )
            s--;
    h.resize(s - 1);
    return h;
}

/* Checks if a point is inside the convex hull: O(log(n))*/

bool inside_triangle(Point a, Point b, Point c, Point point) {
    long long int s1 = abs((b - a).cross(c - b));
    long long int area1 = abs((point - a).cross(point - b));
    long long int area2 = abs((point - b).cross(point - c));
    long long int area3 = abs((point - c).cross(point - a));
    long long int s2 = area1 + area2 + area3;
    return s1 == s2;
}

bool is_inside(vector<Point>& hull, Point p) {
    int n = hull.size();
    if(n == 1) return (hull.front() == p);

    int l = 1, r = n - 1;
    while(abs(r - l) > 1) {
        int mid = (r + l) / 2;
        Point to_mid = hull[mid] - hull[0];
```

```
        Point to_p = p - hull[0];
        if(to_p.cross(to_mid) < 0)
            r = mid;
        else
            l = mid;
    }
    return inside_triangle(hull[0], hull[l], hull[r],
     p);
}
```

3.2 General

```

ld dist  (Point a, Point b){ return sqrtl((a-b)*(a-b)
    ); }                        // distance from A to B
ld angle (Point a, Point b){ return acos((a*b) /
    sqrtl(a*a) / sqrtl(b*b)); } //Angle between A and
     B
Point rotate(Point p, double ang){ return Point(p.x*
    cos(ang) - p.y*sin(ang), p.x*sin(ang) + p.y*cos(
    ang)); } //Left rotation. Angle in radian

ll Area(vector<Point>& p){
  ll area = 0;
  for(int i=2; i < p.size(); i++)
    area += (p[i]-p[0]) % (p[i-1]-p[0]);
  return abs(area) / 2LL;
}

// Intersecao entre duas retas definidas por a1 + td1
     e a2 + td2
// se retas forem paralelas d1 % d2 = 0
Point intersect(Point a1, Point d1, Point a2, Point
    d2){
  return a1 + d1 * (((a2 - a1)%d2) / (d1%d2));
}

ld dist_pt_line(Point a, Point l1, Point l2){
    return abs( ((a-l1) % (l2-l1)) / dist(l1, l2)  );
}

ld dist_pt_segm(Point a, Point s1, Point s2){
  if(s1 == s2) return dist(s1, a);

  Point d = s2 - s1;
  ld t = max(0.0L, min(1.0L, ((a-s1)*d) / sqrtl(d*d))
    );

  return dist(a, s1+(d*t));
}
```

4 Number theory

4.1 Binomial-coefficient

```
/*
    Calcula N escolhe K mod P
*/

ll fact[1000000]; // Preh computar fatoriais
ll comb(ll n, ll k, ll p) {
    return ((fact[n] * inv(fact[k], p) % p) * inv(
    fact[n - k], p)) % p;
}
```

4.2 Modular-inverse

```
/*
Calcula o Inverso Modular de um numero `a` mod `p`
pelo pequeno teorema de fermat.
*/

ll inv(ll a, ll p){
    return fexp(a, p - 2);
}
```

4.3 Utilities

```
// O(sqrt(n))
bool prime(ll a)
{
    if (a == 1)
        return 0;
    for (int i = 2; i <= round(sqrt(a)); ++i)
        if (a % i == 0)
            return 0;
    return 1;
}

// O(log(min(a, b)))
ll gcd(ll a, ll b)
{
    if (!b)
        return a;
    return gcd(b, a % b);
}

// O(log(min(a, b)));
ll lcm(ll a, ll b) {
    return a / gcd(a, b) * b;
}
```

4.4 Sieve-of-erasthotenes

```
/*
Sieve of Erasthotenes
    Consulta rapida de numeros primos
    Complexidade: O(nlog(log(n)))
    Calcula o maior divisor primo de cada numero
*/

bool prime[LIM];
int big_prime[LIM];
void sieve() {
    memset(prime, 1, sizeof prime);
    prime[0] = prime[1] = false;
    for (int i = 2; i < LIM; i++) {
        if (prime[i]) {
            big_prime[i] = i;
            for (int j = i * 2; j < LIM; j += i)
                prime[j] = false, big_prime[j] = i;
        }
    }
}

// Retorna os divisores de `n` O(sqrt(n))
vector<int> divisores(int n)
{
    vector<int> d;
    for (int i = 1; i * i <= n; i++) {
        if (n % i == 0) {
            d.push_back(i);
            if (i != n / i) d.push_back(n / i);
        }
    }
    d.push_back(n);
    return d;
}

// Fatoracao prima de `n` com sieve O(log(n))
vector<int> sieve_factorization(int n) {
    vector<int> primes;
    while (n > 1) {
```

```cpp
        primes.push_back(big_prime[n]);
        n /= big_prime[n];
    }
    return primes;
}

// Fatoracao prima em O(sqrt(n))
vector<pair<int, int>> prime_factorization(int n) {
    vector<pair<int, int>> primes;
    for (int i = 2; i * i <= n; i++) {
        int cnt = 0;
        while (n % i == 0)
            n /= i, cnt++;
        if (cnt > 0)
            primes.push_back({i, cnt});
    }
    if (n > 1)
        primes.push_back({n, 1});
    return primes;
}

// Soma dos divisores de todos os numero de 1 ateh
    LIM - 1
ll sumDivisors[LIM];
void sum_div()
{
    for (int i = 1; i < LIM; i++) {
        for (int j = i; j < LIM; j += i) {
            sumDivisors[j] += i;
        }
    }
}

// Numero dos divisores de todos os numero de 1 ateh
    LIM - 1
ll numDivisors[LIM];
void num_div()
{
    for (int i = 1; i < LIM; i++) {
        for (int j = i; j < LIM; j += i) {
            numDivisors[j]++;
        }
    }
}
```

4.5 Extended-euclidean-algorithm

```cpp
/*
    Algoritmo Estendido de Euclides (Extended GCD)

    Complexidade: O(log(min(a, b)))

    Calcula os coeficientes x e y da equacao
    diofantina:
        ax + by = gcd(a, b)

    Para resolver a equacao ax + by = c, onde c eh um
     valor dado:
    - Primeiro, eh necessario que c % gcd(a, b) == 0.
    - Se sim, as solucÃţes sao:
        x *= c / gcd(a, b)
        y *= c / gcd(a, b)
    - Solucao geral eh
        x(t) = x0 + (b/gcd(a,b)) * t
        y(t) = y0 - (a/gcd(a,b)) * t

*/

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

4.6 Prefix-sum-2d

```cpp
/*
    PrefixSum2D (1-based)
    Calcula queries num subretÃćngulo de um grid:
        - Build - O(nÂš)
        - Queries - O(1)
*/

vector<vector<ll>> pref(maxn, vector<ll>(maxm, 0));
void build(vector<vector<ll>> &grid, int n) {
    // Constroi a PS - O(nÂš)
    for (int i = 1; i <= n; i++) {
        for (int j = 1; j <= n; j++) {
            pref[i][j] = grid[i][j] + pref[i - 1][j]
    + pref[i][j - 1] - pref[i - 1][j - 1];
        }
    }
}

ll query(int pr, int pc, int tr, int tc) {
    return pref[tr][tc] - pref[tr][pc - 1] - pref[pr
    - 1][tc] + pref[pr - 1][pc - 1];
}
```

4.7 Ordered-set

```cpp
/*
    Includes C++ Ordered Set (Lento, pode dar TLE)
    use less_equal pra multiset

    O(log(n))
    * order of key (int n) -  Number of items
    strictly smaller than k.

    O(log(n))
    * find_by_order (int n) - K-th element in a set (
    counting from zero).
*/

#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
using namespace __gnu_pbds;
#define ordered_set tree<int, null_type,less<int>,
    rb_tree_tag,tree_order_statistics_node_update>
```

4.8 Matrix-exponentiation

```cpp
/*
    Exponenciacao Rapida de Matrizes  O(mÂş log (b))
    Calcula recorrÃłncias lineares
*/

int m = 2; // tamanho da matriz
class Matrix{
    public:
        vector<vector<ll>> mat = {{0, 0}, {0, 0}};

        void setSize(int k) {
            m = k;
            mat.assign(m, vector<ll>(m, 0));
        }

        Matrix operator * (const Matrix &p){
```

```
            Matrix ans;
            for(int i = 0; i < m; i++)
                for(int j = 0; j < m; j++)
                    for(int k = ans.mat[i][j] = 0; k < m; k++)
                        ans.mat[i][j] = (ans.mat[i][j] + 1LL * (mat[i][k] % MOD) * (p.mat[k][j] % MOD)) % MOD;
            return ans;
        }
};

// O(log(b))
Matrix fexp(Matrix a, ll b){
    Matrix ans;
    for(int i = 0; i < m; i++)
        ans.mat[i][i] = 1;

    while(b){
        if(b & 1) ans = ans*a;
        a = a*a;
        b >>= 1;
    }
    return ans;
}
```

4.9 Fast-exponentiation

```
/*
    Fast Exponentiation
    Calcula a^b mod m em O(log(n))
*/

ll fexp(ll a, ll b, ll MOD){
    ll ans = 1;
    while(b) {
        if(b & 1) ans = (ans * a) % MOD;
        a = (a * a) % MOD;
        b >>= 1;
    }
    return ans;
}
```

5 Graph

5.1 Dijsktra

```

/*
Dijkstra - Single Source Shortest Path
Complexidade O(n log (n))
*/
vector<ll> dist(maxn, INF);
vector<pii> g[maxn];
vector<ll> dijkstra() {
    priority_queue<pii, vector<pii>, greater<pii>> pq;
    pq.push({0, 0});
    dist[0] = 0;
    while(!pq.empty()) {
        auto [cost, from] = pq.top();
        pq.pop();
        if (dist[from] != cost) continue;
        for (const auto&[w, to]: g[from]) {
            if (dist[from] + w < dist[to]) {
                dist[to] = dist[from] + w;
                pq.push({dist[to], to });
            }
        }
    }
    return dist;
}
```

5.2 Bellman-ford

```
/*
Bellmand Ford Single Source Shortest Path
    Complexidade O(VE)
    Encontra ciclos negativos
*/

struct Edge {
    int from, to, cost;
    Edge(int _f, int _t, int _c): from(_f), to(_t), cost(_c) {}
};
vector<ll> BellmanFord(int n, vector<Edge> &g, int src) {
    vector<ll> distance(n, INF);
    distance[src] = 0;
    for (int u = 0; u < n - 1; u++) {
        for (auto edge : g) {
            auto [from, to, cost] = edge;
            distance[to] = min(distance[to], distance[from] + cost);
        }
    }

    vector<int> negative_cycle(n);
    for (auto edge : g) {
        auto [from, to, cost] = edge;
        if (distance[from] + cost < distance[to]) {
            distance[to] = -INF;
            negative_cycle[to] = true;
        }
    }

    // propaga ciclo negativo e encontra os nos afetados - O(VE)
    for (int u = 0; u < n; u++) {
        if (negative_cycle[u]) {
            queue<int> q;
            q.push(u);
            while (!q.empty()) {
                int node = q.front();
                q.pop();
                for (auto [from, to, cost] : g) {
                    if (from == node && !negative_cycle[to]) {
                        negative_cycle[to] = true;
                        q.push(to);
                    }
                }
            }
        }
    }

    // Marca os nos afetados por ciclos negativos
    for (int i = 0; i < n; i++) {
        if (negative_cycle[i]) {
            distance[i] = -INF;
        }
    }

    return distance;
}
```

5.3 Floyd-warshall

```
/*
    Floyd Warshall - All Pairs Shortest Path
    Funciona apenas em matrizes
    Complexidade O(nÂş)
*/

```

```cpp
vector<vector<ll>> FloydWarshall(int n, vector<vector<int>> &graph) {
    // precomputa distÃćncias O(nÂš)
    vector<vector<ll>> distance(n, vector<ll>(n, INF));
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            if (i == j) {
            distance[i][j] = 0;
            } else if (graph[i][j] != -1) {
                distance[i][j] = graph[i][j];
            }
        }
    }

    // O(nÂş)
    for (int k = 0; k < n; k++) {
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < n; j++) {
                distance[i][j] = min(distance[i][j], distance[i][k] + distance[k][j]);
            }
        }
    }
}
```

5.4 Kahn

```cpp
/*
    Kahn Topological Sorting
    Complexidade - O(V + E)

    Encontra a ordenacao topologica e detecta ciclos ao
    mesmo tempo
*/
vector<int> KahnToposort(int n, vector<int> *graph) {
    vector<int> in_degree(n);
    for (int i = 0; i < n; i++) {
        for (int to : graph[i]) {
            in_degree[to]++;
        }
    }
    queue<int> q;
    for (int i = 0; i < n; i++) {
        if (in_degree[i] == 0)
            q.push(i);
    }

    int idx = 0;
    vector<int> order(n);
    while (!q.empty()) {
        int u = q.front(); q.pop();
        order[idx++] = u;
        for (int v: graph[u]) {
            in_degree[v]--;
            if (in_degree[v] == 0) {
                q.push(v);
            }
        }
    }

    if (idx != n) {
        return {};  // cycle detected
    }

    return order;
}
```

5.5 Kruskal

```cpp
struct Edge
{
    int u, v, w;
    Edge() {}
    Edge(int a, int b, int c): u(a), v(b), w(c) {}
    bool operator<(const Edge &s) const { return w < s.w; }
};

/*
    Encontra o custo da Arvore Geradora Minima
    Complexidade O(E log E)
    find(u) e unite(u, v) de Union-Find
*/

ll Kruskal(vector<Edge> &g) {
    sort(begin(g), end(g));
    ll total = 0;
    for (auto [u, v, w]: g) {
        if (find(u) != find(v)) {
            unite(u, v);
            total += w;
        }
    }
    return total;
}
```

6 String

6.1 Double-hash

```cpp
/*
    Double Polynomial Hashing
    Prehcalculo - O(n)
    Substring hash queries - O(1)
    Hash(l, m - 1) calcula o hash da substring
    incluindo o l de tamanho m
*/

const int MOD1 = 188'888'881;
const int MOD2 = 1e9 + 7;
const int base = 137;


ll pow1[MAXN];
ll pow2[MAXN];

// O(n) - Chamar antes
void calc_pow()
{
    pow1[0] = pow2[0] = 1;
    for (int i = 1; i < MAXN; i++)
        pow1[i] = (pow1[i - 1] * base) % MOD1,
        pow2[i] = (pow2[i - 1] * base) % MOD2;
}

struct Hashing
{
    vector<pair<ll, ll>> pref;
    // O(1)
    Hashing(string &s)
    {
        pref = vector<pair<ll, ll>>(s.size() + 1, {0, 0});
        for (int i = 0; i < s.size(); i++)
            pref[i + 1].first = ((pref[i].first * base) % MOD1 + s[i]) % MOD1,
            pref[i + 1].second = ((pref[i].second * base) % MOD2 + s[i]) % MOD2;
    }

    // O(1)
    ll operator()(int a, int b)
```

```
    {
        ll h1 = (MOD1 + pref[b + 1].first - (pref[a].first * pow1[b - a + 1]) % MOD1) % MOD1;
        ll h2 = (MOD2 + pref[b + 1].second - (pref[a].second * pow2[b - a + 1]) % MOD2) % MOD2;
        return (h1 << 32) | h2;
    }
};
```

6.2 Trie

```
const int ALPHA = 26; // tamanho do alfabeto
/*
    Trie - arvore de Prefixos
    maxn - Soma do tamanho de todas as strings
*/
int trie[maxn][ALPHA], word_end[maxn], z = 1;

// Add(P) - O(|P|)
void add(string &s) {
    int cur = 0;
    for(int i = 0; i < s.size(); i++) {
        if (trie[cur][s[i] - 'a'] == -1) {
            memset(trie[z], -1, sizeof trie[z]);
            trie[cur][s[i] - 'a'] = z++;
        }
        cur = trie[cur][s[i] - 'a'];
    }
    word_end[cur]++;
}

// Query(P) - O(|P|)
int query(string &s){
    int cur = 0;
    for(int i = 0; i < s.size(); i++){
        if(trie[cur][s[i] - 'a'] == -1) return 0;
        cur = trie[cur][s[i] - 'a'];
    }
    return word_end[cur];
}

// Sempre inicializar antes
void init(){
    memset(trie[0], -1, sizeof trie[0]);
    memset(word_end, 0, sizeof word_end);
    z = 1;
}
```

7 Data structures

7.1 Rmq

```
/*
Sparse Table RMQ Range Min/Max Query

Build O(n log n)
Query O(1)
*/

const int MAXN = 1e5 + 5;
const int MAXLG = 31 - __builtin_clz(MAXN) + 1;

int value[MAXN], table[MAXLG][MAXN];

void build(int N){
    for(int i=0; i<N; i++) table[0][i] = value[i];

    for(int p=1; p < MAXLG; p++)
        for(int i=0; i + (1 << p) <= N; i++)
            table[p][i] = min(table[p-1][i], table[p-1][i+(1 << (p-1))]); // ou max
}

int query(int l, int r){
    int p = 31 - __builtin_clz(r - l + 1);  //floor log
    return min(table[p][l], table[p][ r - (1<<p) + 1]);
}
```

7.2 Fenwick-tree

```
/*
    Fenwick Tree - Range Queries
*/

vector<int> bit(maxn);
int n; // tamanho do array 0-based

// O(log(n))
void add(int pos, int val) {
    ++pos;
    while (pos <= n) {
        bit[pos] += val;
        pos += (pos & (-pos));
    }
}

// O(log(n))
int query(int pos) {
    ++pos;
    int sum = 0;
    while(pos > 0) {
        sum += bit[pos];
        pos -= (pos & (-pos));
    }
    return sum;
}
```

7.3 Union-find

```
/*
    Disjoint Set Union with path compression
    Complexidade:
        - find(u) O(alpha(n))
        - unite(u) O(alpha(n))
*/

const int MAXN = 2e5 +5;
struct UnionFind {
    int parents[MAXN];
    int sizes[MAXN];

    // O(n)
    void init(int n) {
        for (int i = 1; i <= n; i++) {
            parents[i] = i;
            sizes[i] = 1;
        }
    }

    // O(alpha(n)) ~ O(1)
    int find(int x) { return parents[x] == x ? x : (parents[x] = find(parents[x])); }

    // O(alpha(n)) ~ O(1)
    bool unite(int x, int y) {
        int x_root = find(x);
        int y_root = find(y);
        if (x_root == y_root) { return false; }
        if (sizes[x_root] < sizes[y_root]) { swap(x_root, y_root); }
        sizes[x_root] += sizes[y_root];
```

```cpp
        parents[y_root] = x_root;
        return true; // (some condition met for component);
    }
};
```

7.4 Segment-tree

```cpp
// 1-Based Segment Tree - Range Queries
const int MAXN = 2e5 + 5;

int n; // numero de nodes
vector<int> a(MAXN); // vetor de input 1-based

struct SegmentTree {
    vector<int> tree;
    SegmentTree() {
        tree.resize(4 * (n + 1));
    }

    // Join - Funcao a rodar nos nos da arvore, min, max, etc.
    int join(int a, int b) {
    }

    // O(n)
    void build(int l = 1, int r = n, int v = 1) {
        if (l == r) {
            tree[v] = a[l];
            return;
        } else {
            int mid = l + (r - l) / 2;
            build(l, mid, v * 2);
            build(mid + 1, r, v * 2 + 1);
            tree[v] = join(tree[v * 2], tree[v * 2 + 1]);
        }
    }

    // O(log(n))
    void update(int pos, int val, int l = 1, int r = n, int v = 1) {
        if (l == r) {
            tree[v] = val;
            return;
        } else {
            int mid = l + (r - l) / 2;
            if (pos <= mid) {
                update(pos, val, l, mid, v * 2);
            } else {
                update(pos, val, mid + 1, r, v * 2 + 1);
            }
            tree[v] = join(tree[v * 2], tree[v * 2 + 1]);
        }
    }

    // O(log(n))
    ll query(int a, int b, int l = 1, int r = n, int v = 1) {
        if (b < l || a > r) return (1e9 + 9);
        if (a <= l && r <= b) return tree[v];
        int mid = l + (r - l) / 2;
        ll left = query(a, b, l, mid, v * 2);
        ll right = query(a, b, mid + 1, r, v * 2 + 1);
        return join(left, right);
    }
};
```