Contents

Segtrio - Algoritmos

Gerado em Fri Sep 12 13:01:21 -03 2025

Este documento consolida todos os arquivos .cpp do projeto com índice.

Combinatorics/pie.cpp

Caminho: `Combinatorics/pie.cpp`

```cpp
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
typedef vector<ll> v64;
#define forn(i, s, e) for (ll i = (s); i < (e); i++)

// conta quantos números de 1 a r são divisíveis por pelo menos um
// dos primos em p
ll count_multiples(ll r, v64& p) {
    ll k = p.size();
    ll sum = 0;
    for (ll mask = 1; mask < (1 << k); mask++) {
        ll mult = 1, sig = -1;
        forn (i, 0, k) {
            sig *= (mask & (1 << i)) ? -1 : 1;
            mult *= (mask & (1 << i)) ? p[i] : 1;
        }
        ll cur = r / mult;
        sum += sig * cur;
    }
    return sum;
}
```

DP/SubsetSumOptimization.cpp

Caminho: `DP/SubsetSumOptimization.cpp`

```
// subset sum optimization: O( (k*lim) + (k/64 * (k/lim)) )
// k é a soma máxima dos valores que se pode/quer encontrar.
// com lim = sqrt(k) fica O(ksqrt(k)). lim em geral pode ser um pouco menor pelo bitset dividindo por 6

v64 v;
ll lim = 100;
set<ll> big;
set<ll> small;
v64 cnt(n+1, 0);
bitset<1000001> b;
b[0] = 1;

for(auto val : v){
    if(sz >= lim) big.insert(sz);
    else small.insert(sz);
    cnt[sz] += 1;
}

for(auto sz : small){ // lim * k
    forn(i,0,sz){
        dbg(i);
        ll ct = 0;
        for(ll j = i; j <= k; j += sz){
            dbg(b[j]);
            if(b[j]) ct = cnt[sz];
            else if(ct){
                b[j] = 1;
                ct--;
            }
        }
    }
}

for(auto sz : big){ // k * (k/lim)/64 operacoes
    ll ct = min(cnt[sz], k/sz);
    forn(i,0,ct) b |= (b << sz);
}
```

DP/bitmask.cpp

Caminho: `DP/bitmask.cpp`

```
// Dicas base para uso de DP Bitmask. Iteração por cada mask: O(2^n).
// Contar quantos já foram na dp: __builtin_popcount(mask);
```

```cpp
ll sz = 1<<n;
v64 dp(sz); //dp padrão tamanho 2^n
dp[0] = 1 //estado inicial

forn(mask,0,sz){ // fazer algo a cada mask. Exemplo (hamiltonian flights):
    forn(i,0,n){
        if (!(mask & (1<<i))) continue;
        forn(j,0,n){
            if((mask & (1 << j))) continue;
            ll nmask = mask | (1 << j);
            // faz algo como:
            dp[nmask] = max(dp[nmask], dp[mask] + 1);
        }
    }
}
```

Data Structures/BIT/PointQueryRangeUpdate.cpp

Caminho: `Data Structures/BIT/PointQueryRangeUpdate.cpp`

```cpp
struct BIT { // indexada a 1, range update point sum. O(N) espaco e construcao
    v64 bit;
    ll n;

    BIT(ll sz){
        n = sz + 1;
        bit.assign(n, 0);
    }

    BIT(const v64& v) : BIT(v.size()) {
        v64 delta(n-1);
        delta[0] = v[0];
        forn(i, 1, n-1)
            delta[i] = v[i] - v[i - 1];

        forn(i, 1, n) {
            bit[i] += delta[i-1];
            ll j = i + (i & -i);
            if (j < n)
                bit[j] += bit[i];
        }
    }

    ll point_query(ll i){
        ll sum = 0;
        for(++i; i > 0; i -= i & -i)
            sum += bit[i];
        return sum;
    }

    void add(ll i, ll delta){
        for(++i; i < n; i += i & -i){
            bit[i] += delta;
        }
    }

    void range_add(ll l, ll r, ll val) {
        add(l, val);
        add(r + 1, -val);
    }
};
```

Data Structures/BIT/RangeMinPointUpdate.cpp

Caminho: `Data Structures/BIT/RangeMinPointUpdate.cpp`

```cpp
// range min point update;
// limitations:
//   can only answer queries of type [0, r]
//  the new value updated has to be smaller than the current value
// solution for this: Efficient Range Minimum Queries using Binary Indexed Trees
struct BIT {
    v64 bit;
    ll n;

    BIT(ll n) {
        this->n = n;
        bit.assign(n, INF);
    }

    BIT(v64 a) : BIT(a.size()) {
        forn(i,0,n)
            update(i, a[i]);
    }

    ll getmin(ll r) {
        ll ret = INF;
        for (; r >= 0; r = (r & (r + 1)) - 1)
            ret = min(ret, bit[r]);
        return ret;
    }

    void update(ll idx, ll val) {
        for (; idx < n; idx = idx | (idx + 1))
            bit[idx] = min(bit[idx], val);
    }
};
```

Data Structures/BIT/RangeSumPointUpdate.cpp

Caminho: `Data Structures/BIT/RangeSumPointUpdate.cpp`

```cpp
// indexada a 1, range sum point update.
// O(n) espaco e construcao
// O(log(n)) queries

struct BIT {
    v64 bit;
    ll n;

    BIT(ll sz){
        n = sz + 1;
        bit.assign(n, 0);
    }

    BIT(const v64& v) : BIT(v.size()) {
        forn(i,1,n){
            bit[i] += v[i-1];
            ll j = i + (i & -i);
            if(j < n)
                bit[j] += bit[i];
        }
    }

    ll prefSum(ll i){
        ll sum = 0;
        for(++i; i > 0; i -= i & -i)
            sum += bit[i];
        return sum;
    }

    ll query(ll a, ll b){
        return prefSum(b) - prefSum(a-1);
    }

    void add(ll i, ll delta){
        for(++i; i < n; i += i & -i){
            bit[i] += delta;
        }
    }

    void set(ll i, ll val){
        add(i, val - query(i,i));
    }
};
```

Data Structures/Segtree/segLazy.cpp

Caminho: `Data Structures/Segtree/segLazy.cpp`

```
// Segment Tree (Range Query + Point Update)
//
// Balanced binary tree for range queries with a customizable combine; supports point updates and range
//
// complexity: O(log N) per op, O(N)

struct lazy {
    ll add = 0;
    optional<ll> set;

    void compose(const lazy& o) {
        if (o.set.has_value()) {
            set = o.set;
            add = 0;
        }

        if (o.add != 0) {
            if (set.has_value()) *set += o.add;
            else add += o.add;
        }
    }
};

struct node {
    ll val = 0;

    static node comb(const node& a, const node& b) {
        return {min(a.val, b.val)};
    }

    void resolve(const lazy& lz, ll l, ll r) {
        if (lz.set.has_value()) val = *lz.set;
        if (lz.add) val += lz.add;
    }
};

const node neutral = {INF};

struct tree {
    ll lm, rm;
    unique_ptr<tree> lc, rc;

    node val;
    lazy lz;

    tree(ll l_, ll r_, const vector<node>& v) : lm(l_), rm(r_) {
        if (lm == rm) val = v[lm];
        else {
            ll m = (lm + rm) / 2;
            lc = make_unique<tree>(lm, m, v);
            rc = make_unique<tree>(m + 1, rm, v);
            pull();
        }
    }

    void pull() {
        val = node::comb(lc->val, rc->val);
    }

    void push() {
        val.resolve(lz, lm, rm);
        if (lm != rm) {
            lc->lz.compose(lz);
            rc->lz.compose(lz);
        }
        lz = {};
    }

    void range_update(ll lq, ll rq, lazy x) {
        push();
        if (rq < lm || lq > rm) return;
        if (lq <= lm && rm <= rq) {
            lz.compose(x);
            push();
            return;
        }
        lc->range_update(lq, rq, x);
        rc->range_update(lq, rq, x);
        pull();
    }

    node query(ll lq, ll rq) {
        push();
        if (rq < lm || lq > rm) return neutral;
        if (lq <= lm && rm <= rq) return val;
        return node::comb(lc->query(lq, rq), rc->query(lq, rq));
    }
};
```

Data Structures/dsu.cpp

Caminho: Data Structures/dsu.cpp

```
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
typedef vector<ll> v64;

struct dsu {
    v64 id, sz;

    dsu(ll n) : id(n), sz(n, 1) { iota(id.begin(), id.end(), 0); }

    ll find(ll a) { return id[a] == a ? a : id[a] = find(id[a]); }

    void uni(ll a, ll b) {
        a = find(a), b = find(b);
        if (a == b) return;
        if (sz[a] < sz[b]) swap(a, b);
        sz[a] += sz[b], id[b] = a;
    }
};
```

Data Structures/mo.cpp

Caminho: Data Structures/mo.cpp

```
// Mo Algorithm: O((N + Q) * sqrt(N))

ll len;
struct Query {
    ll l, r, i, k;
    bool operator<(Query other) const {
        ll bl = l/len;
        ll o_bl = other.l/len;
        if(bl != o_bl) return bl < o_bl;
        if(bl & 1) return r > other.r;
        else return r < other.r;
    }
    Query(ll l_, ll r_, ll i_, ll k_) : l(l_), r(r_), i(i_), k(k_) {}
};


inline void remove(ll idx){}
inline void add(ll idx) {}

void solve() {
    len = sqrt(n);

    vector<Query> queries;
    forn(i,0,m){
        ll v, k; cin >> v >> k;
        v--;
        queries.push_back(Query(ranges[v].first,ranges[v].second, i, k));
    }
    sort(queries.begin(), queries.end());

    v64 ans(m);
    for(auto q : queries){
        while(cl > q.l) add(--cl);
        while(cr < q.r) add(++cr);
        while(cl < q.l) remove(cl++);
        while(cr > q.r) remove(cr--);
        ans[q.i] = //logica
    }
    forn(i,0,m) cout << ans[i] << ln;
}
```

Data Structures/sqrt.cpp

Caminho: `Data Structures/sqrt.cpp`

```
ll n;
ll len;
v64 v;
v64 b;
v64 lazySum;
v64 lazySet;

void lazy_down(ll blk){
    forn(i,blk*len,(blk+1)*len){
        if(lazySet[blk] != -1){}
        if(lazySum[blk] != 0){}
    }
    lazySum[blk] = 0;
    lazySet[blk] = -1;
}

void solve() {
    len = sqrt(n) + 1;
    b.assign(len, 0);
    lazySum.assign(len, 0);
    lazySet.assign(len, -1);

    forn(i,0,n) cin >> v[i];
    forn(i,0,n) b[i/len] += v[i];

    // queries:
    ll l, r;
    ll bl = l/len;
    ll br = r/len;
    ll sum = 0;
    if(bl == br){
        lazy_down(bl);
        forn(i,l,r+1) {}
    } else {
        lazy_down(bl);
        lazy_down(br);
        forn(i,l,(bl+1)*len) {}
        forn(i,bl+1, br) {}
        forn(i,br*len, r+1) {}
    }
}
```

Geometry/notes.cpp

Caminho: `Geometry/notes.cpp`

```
// === GEOMETRY: Orientação LEFT/RIGHT/TOUCH e primitivas cross/diff ===
long long getCrossProduct(const v64& v1, const v64& v2){return v1[0]*v2[1]-v1[1]*v2[0];}
v64 getDifference(const v64& v1, const v64& v2){vector<long long> d(2); d[0]=v2[0]-v1[0]; d[1]=v2[1]-v1
void solve_orient(){ ll x1,y1,x2,y2,x3,y3; cin>>x1>>y1>>x2>>y2>>x3>>y3;
v64 p1={x1,y1},p2={x2,y2},p3={x3,y3}; v64 a=getDifference(p1,p2), b=getDifference(p1,p3);
ll s=getCrossProduct(a,b); if(s>0) cout<<"LEFT\n"; else if(s<0) cout<<"RIGHT\n"; else cout<<"TOUCH\n";}
// === GEOMETRY: Distância entre dois pontos ===
long double getDist(const v64& diff){ return hypotl((long double)diff[0], (long double)diff[1]); }
void solve_dist(){ ll x1,y1,x2,y2; cin>>x1>>y1>>x2>>y2; v64 d=getDifference({x1,y1},{x2,y2}); cout<<setp
// === GEOMETRY: Estruturas auxiliares ===
struct Pole { ll x,y; int z; long double ang; long long r2; };
long double polar_angle(ll x,ll y){ long double t=atan2l((long double)y,(long double)x); if(t<0) t+=2.0
long double angle_between(ll ax,ll ay,ll bx,ll by){
long double da=hypotl((long double)ax,(long double)ay);
long double db=hypotl((long double)bx,(long double)by);
long double dot=(long double)ax*bx+(long double)ay*by;
long double c=dot/(da*db); if(c>1)c=1; if(c<-1)c=-1; return acosl(c);
}
long long power_of_point(ll x1,ll y1,ll r,ll x2,ll y2){ ll dx=x2-x1, dy=y2-y1; return dx*dx+dy*dy - r*r
// === GEOMETRY: Ordenação CCW a partir do semieixo x<=0 ===
typedef long double ld;
const ld eps = 1e-12L;
```

```
inline bool eq(ld a, ld b){ return fabsl(a-b)<=eps; }
struct pt{ ld x,y; pt(ld x_=0,ld y_=0):x(x_),y(y_){} ld operator^(const pt& p)const{return x*p.y - y*p.
inline ld dist2(pt p, pt q){ ld dx=p.x-q.x,dy=p.y-q.y; return dx*dx+dy*dy; }
inline int half(const pt& p){ if(p.y<0) return 0; if(p.y>0) return 1; return (p.x<=0)?0:1; }
inline bool angCmp(const pt& a,const pt& b){int ha=half(a), hb=half(b); if(ha!=hb) return ha<hb; ld cr=
```

Graph Theory/dijkstra.cpp

Caminho: Graph Theory/dijkstra.cpp

```
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
typedef pair<ll, ll> p64;
typedef vector<ll> v64;
#define forn(i, s, e) for (ll i = (s); i < (e); i++)
const ll INF = 0x3f3f3f3f3f3f3f3fll;

vector<vector<p64>> adj;

void dijkstra(ll s, v64& d, v64& p) {
    ll n = adj.size();
    d.assign(n, INF);
    p.assign(n, -1);

    d[s] = 0;
    priority_queue<p64, vector<p64>, greater<p64>> q;
    q.push({0, s});
    while (!q.empty()) {
        ll v = q.top().second;
        ll d_v = q.top().first;
        q.pop();
        if (d_v != d[v]) continue;

        for (auto edge : adj[v]) {
            ll to = edge.first;
            ll len = edge.second;

            if (d[v] + len < d[to]) {
                d[to] = d[v] + len;
                p[to] = v;
                q.push({d[to], to});
            }
        }
    }
}
```

Graph Theory/dinitz.cpp

Caminho: Graph Theory/dinitz.cpp

```
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
const ll INF = 0x3f3f3f3f3f3f3f3fll;
```

```
struct dinitz {
    const bool scaling = false; // com scaling -> O(nm log(MAXCAP)),
    int lim;                    // com constante alta
    struct edge {
        int to, cap, rev, flow;
        bool res;
        edge(int to_, int cap_, int rev_, bool res_)
            : to(to_), cap(cap_), rev(rev_), flow(0), res(res_) {}
    };

    vector<vector<edge>> g;
    vector<int> lev, beg;
    ll F;
    dinitz(int n) : g(n), F(0) {}

    void add(int a, int b, int c) {
        g[a].emplace_back(b, c, g[b].size(), false);
        g[b].emplace_back(a, 0, g[a].size()-1, true);
    }
    bool bfs(int s, int t) {
        lev = vector<int>(g.size(), -1); lev[s] = 0;
        beg = vector<int>(g.size(), 0);
        queue<int> q; q.push(s);
        while (q.size()) {
            int u = q.front(); q.pop();
            for (auto& i : g[u]) {
                if (lev[i.to] != -1 or (i.flow == i.cap)) continue;
                if (scaling and i.cap - i.flow < lim) continue;
                lev[i.to] = lev[u] + 1;
                q.push(i.to);
            }
        }
        return lev[t] != -1;
    }
    int dfs(int v, int s, int f = INF) {
        if (!f or v == s) return f;
        for (int& i = beg[v]; i < g[v].size(); i++) {
            auto& e = g[v][i];
            if (lev[e.to] != lev[v] + 1) continue;
            int foi = dfs(e.to, s, min(f, e.cap - e.flow));
            if (!foi) continue;
            e.flow += foi, g[e.to][e.rev].flow -= foi;
            return foi;
        }
        return 0;
    }
    ll max_flow(int s, int t) {
        for (lim = scaling ? (1<<30) : 1; lim; lim /= 2)
            while (bfs(s, t)) while (int ff = dfs(s, t)) F += ff;
        return F;
    }
};
```

```
// Recupera as arestas do corte s-t
vector<pair<int, int>> get_cut(dinitz& g, int s, int t) {
    g.max_flow(s, t);
    vector<pair<int, int>> cut;
    vector<int> vis(g.g.size(), 0), st = {s};
    vis[s] = 1;
    while (st.size()) {
        int u = st.back(); st.pop_back();
        for (auto e : g.g[u]) if (!vis[e.to] and e.flow < e.cap)
            vis[e.to] = 1, st.push_back(e.to);
    }
    for (int i = 0; i < g.g.size(); i++) for (auto e : g.g[i])
        if (vis[i] and !vis[e.to] and !e.res) cut.emplace_back(i, e.to);
    return cut;
}
```

Graph Theory/edmonds_karp.cpp

Caminho: Graph Theory/edmonds_karp.cpp

```
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
typedef pair<ll, ll> p64;
typedef vector<ll> v64;
const ll INF = 0x3f3f3f3f3f3f3f3fll;

ll n;
vector<v64> capacity;
vector<v64> adj;
vector<p64> min_cut;

ll bfs(ll s, ll t, v64& parent) {
    fill(parent.begin(), parent.end(), -1);
    parent[s] = -2;
    queue<p64> q;
    q.push({s, INF});

    while (!q.empty()) {
        ll cur = q.front().first;
        ll flow = q.front().second;
        q.pop();

        for (ll next : adj[cur]) {
            if (parent[next] == -1 && capacity[cur][next]) {
                parent[next] = cur;
                ll new_flow = min(flow, capacity[cur][next]);
                if (next == t) return new_flow;
                q.push({next, new_flow});
            }
        }
    }

    return 0;
}
```

```
ll maxflow(ll s, ll t) {
    ll flow = 0;
    v64 parent(n);
    ll new_flow;

    while (new_flow = bfs(s, t, parent)) {
        flow += new_flow;
        ll cur = t;
        while (cur != s) {
            ll prev = parent[cur];
            capacity[prev][cur] -= new_flow;
            capacity[cur][prev] += new_flow;
            cur = prev;
        }
    }

    return flow;
}

void dfs(ll u, v64& id) {
    id[u] = 1;
    for (ll v : adj[u]) {
        if (id[v] == 1) continue;
        if (!capacity[u][v]) {
            min_cut.push_back({u, v});
            continue;
        }
        dfs(v, id);
    }
}

void get_cut(ll s, ll t) {
    v64 id(n, 0);
    dfs(s, id);
}
```

Graph Theory/kosaraju.cpp

Caminho: Graph Theory/kosaraju.cpp

```
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
typedef vector<ll> v64;
#define forn(i, s, e) for (ll i = (s); i < (e); i++)

vector<bool> vis;

void dfs(ll u, vector<v64> const& adj, v64& out) {
    vis[u] = true;
    for (auto v : adj[u]) {
        if (vis[v]) continue;
        dfs(v, adj, out);
    }
    out.push_back(u);
}

void kosaraju(vector<v64>& adj, vector<v64>& scc, vector<v64>& adj_cond) {
    ll n = adj.size();
    scc.clear(), adj_cond.clear();
    v64 order;
    vis.assign(n, false);

    forn (u, 0, n) {
        if (!vis[u]) dfs(u, adj, order);
    }

    vector<v64> adj_rev(n);
    forn (u, 0, n) {
        for (auto v : adj[u]) {
            adj_rev[v].push_back(u);
        }
    }

    vis.assign(n, false);

    reverse(order.begin(), order.end());

    v64 roots(n, 0);

    for (auto u : order) {
        if (!vis[u]) {
            v64 component;
            dfs(u, adj_rev, component);
            scc.push_back(component);
            ll root = *min_element(component.begin(), component.end());
            for (auto v : component) roots[v] = root;
        }
    }

    adj_cond.assign(n, {});
    forn (u, 0, n) {
        for (auto v : adj[u]) {
            if (roots[u] != roots[v]) adj_cond[roots[u]].push_back(roots[v]);
        }
    }
}
```

Graph Theory/kruskal.cpp

Caminho: Graph Theory/kruskal.cpp

```
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
typedef pair<ll, ll> p64;
typedef vector<ll> v64;
#define forn(i, s, e) for (ll i = (s); i < (e); i++)
#define ln "\n"
```

```
const ll INF = 0x3f3f3f3f3f3f3f3fll;

struct Edge {
    int u, v, weight;
    bool operator<(Edge const& other) {
        return weight < other.weight;
    }
};

vector<Edge> kruskal(ll n, vector<Edge>& edges) {
    ll cost = 0;
    v64 tree_id(n);
    vector<Edge> result;
    forn (i, 0, n) tree_id[i] = i;

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

    for (Edge e : edges) {
        if (tree_id[e.u] != tree_id[e.v]) {
            cost += e.weight;
            result.push_back(e);

            ll old_id = tree_id[e.u], new_id = tree_id[e.v];
            forn (i, 0, n) {
                if (tree_id[i] == old_id) tree_id[i] = new_id;
            }
        }
    }

    return result;
}
```

Graph Theory/lca.cpp

Caminho: Graph Theory/lca.cpp

```
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
typedef vector<ll> v64;
#define forn(i, s, e) for (ll i = (s); i < (e); i++)

ll n, l, timer;
vector<v64> t, up;
v64 tin, tout;

void dfs(ll u, ll p) {
    tin[u] = ++timer;
    up[u][0] = p;
    forn (i, 1, l + 1) {
        up[u][i] = up[up[u][i - 1]][i - 1];
    }

    for (auto v : t[u]) {
        if (v == p) continue;
        dfs(v, u);
    }
    tout[u] = ++timer;
}

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

ll lca(ll u, ll v) {
    if (is_ancestor(u, v)) return u;
    if (is_ancestor(v, u)) return v;
    for (ll i = l; i >= 0; i--) {
        if (!is_ancestor(up[u][i], v)) u = up[u][i];
    }
    return up[u][0];
}

void preprocess(ll root) {
    tin.resize(n);
    tout.resize(n);
    timer = 0;
    l = ceil(log2(n));
    up.assign(n, v64(l + 1));
    dfs(root, root);
}
```

Graph Theory/prim.cpp

Caminho: `Graph Theory/prim.cpp`

```
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
typedef pair<ll, ll> p64;
typedef vector<ll> v64;
#define forn(i, s, e) for (ll i = (s); i < (e); i++)
#define ln "\n"
const ll INF = 0x3f3f3f3f3f3f3f3fll;

struct Edge {
    ll w = INF, to = -1;
    bool operator<(Edge const& other) const {
        return make_pair(w, to) < make_pair(other.w, other.to);
    }
};

ll n;
vector<vector<Edge>> adj;

void prim() {
    ll total_weight = 0;
    vector<Edge> min_e(n);
    min_e[0].w = 0;
    set<Edge> q;
    q.insert({0, 0});
    vector<bool> selected(n, false);
    forn (i, 0, n) {
        if (q.empty()) {
            cout << "No MST!" << ln;
            exit(0);
        }

        ll v = q.begin()->to;
        selected[v] = true;
        total_weight += q.begin()->w;
        q.erase(q.begin());

        if (min_e[v].to != -1)
            cout << v << " " << min_e[v].to << ln;

        for (Edge e : adj[v]) {
            if (!selected[e.to] && e.w < min_e[e.to].w) {
                q.erase({min_e[e.to].w, e.to});
                min_e[e.to] = {e.w, v};
                q.insert({e.w, e.to});
            }
        }
    }

    cout << total_weight << ln;
}
```

Graph Theory/topsort.cpp

Caminho: Graph Theory/topsort.cpp

```
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
typedef vector<ll> v64;
#define forn(i, s, e) for (ll i = (s); i < (e); i++)

ll n;
vector<v64> g;
v64 vis, top;
bool cycle = false;

void dfs(ll u) {
    vis[u] = 1;
    for (auto v : g[u]) {
        if (vis[v] == 1) {
            cycle = true;
            continue;
        }
        if (vis[v] == 2) continue;
        dfs(v);
    }
    vis[u] = 2;
    top.push_back(u);
}

void topsort() {
    vis.assign(n, false);
    top.clear();
    forn (u, 0, n) {
        if (!vis[u]) dfs(u);
    }
    reverse(top.begin(), top.end());
}

bool kahn() {
    top.resize(n);
    v64 indeg(n);
    forn (u, 0, n) {
        for (auto v : g[u]) indeg[v]++;
    }
    queue<ll> q;
    forn (u, 0, n) {
        if (indeg[u] == 0) q.push(u);
    }

    ll idx = 0;
    while (!q.empty()) {
        ll at = q.front(); q.pop();
        top[idx++] = at;
        for (auto to : g[at]) {
            indeg[to]--;
            if (indeg[to] == 0) q.push(to);
        }
    }
    if (idx != n) return false;
    return true;
}
```

Math/sieveFi.cpp

Caminho: `Math/sieveFi.cpp`

```
// Faz o sieve calculando a função fi. O(N*log(N)).
struct sieveCell {
    ll maxDiv;
    ll fi;
    set<ll> distinctPrimes;
    sieveCell(ll max, ll f){maxDiv = max; fi = f; distinctPrimes = {};}
};

vector<sieveCell> makeSieve(){
    vector<sieveCell> sv(N+1, sieveCell(-INF, -INF));
    v64 primes;
    forn(i, 2, N+1){
        if(sieve[i].maxDiv == -INF){
            primes.push_back(i);
            sv[i].maxDiv = i;
            sv[i].fi = i-1;
            sv[i].distinctPrimes = {i};
        }
        for(ll prime : primes){
            if(prime > sv[i].maxDiv) break;
            if(i*prime > N) break;
            sv[i*prime].maxDiv = i;
            sv[i*prime].distinctPrimes = sv[i].distinctPrimes;
            sv[i*prime].distinctPrimes.insert(prime);
            sv[i*prime].fi = i*prime;
            for(ll distPrime : sv[i*prime].distinctPrimes){
                sv[i*prime].fi = (sv[i*prime].fi - sv[i*prime].fi/distPrime);
            }
        }
    }
    return sv;
}
```

Math/singlePrimeFactors.cpp

Caminho: `Math/singlePrimeFactors.cpp`

```
// calcula fatores primos de um número em O(sqrt(N))

map<ll,ll> calcPrimeFactors(ll n) {
    map<ll,ll> pfact;
    ll factor = 0;
    ll pot = 0;
    for (ll i = 2; i*i <= n; i++) {
        while (n % i == 0) {
            factor = i;
            pot++;
            n /= i;
        }
        if(pot > 0){
            pfact[factor] = pot;
            pot = 0;
        }
    }
    if (n > 1) pfact[n] = 1;
    return pfact;
}
```

Miscellaneous/permutations.cpp

Caminho: `Miscellaneous/permutations.cpp`

```
// Roda código a cada permutação dos valores de um vetor. Não gera permutações repetidas.
// O(N!) amortizado se com valores distintos; provável O(N*N!) com valores repetidos.

v64 v;
sort(v.begin(), v.end()); // garante ordem inicial. Precisa ser não decrescente se não vai gerar apenas
do
{
    // não usar nada com overhead (set, map...)
}
```

```
while (next_permutation(v.begin(), v.end()));
```

Miscellaneous/ternary_search.cpp

Caminho: `Miscellaneous/ternary_search.cpp`

```
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
typedef vector<ll> v64;
#define forn(i, s, e) for (ll i = (s); i < (e); i++)

double f(ll x) { return 0.0; }

double ternary_search(double l, double r) {
    double eps = 1e-9;              //set the error limit here
    while (r - l > eps) {
        double m1 = l + (r - l) / 3;
        double m2 = r - (r - l) / 3;
        double f1 = f(m1);      //evaluates the function at m1
        double f2 = f(m2);      //evaluates the function at m2
        if (f1 < f2)
            l = m1;
        else
            r = m2;
    }
    return f(l);                    //return the maximum of f(x) in [l, r]
}
```

Strings/hash.cpp

Caminho: `Strings/hash.cpp`

```
/*
STRINGS/HASH - Polynomial Rolling Hash (double hash)
O que é: representa uma string s[0..n-1] por H = sum(s[i]*B^i) mod M.
Para substring s[l..r] usa H(l,r) = (pref[r] - pref[l-1]) * invB^l (mod M).
Escolhas seguras:
- Bases (B): 911382323, 972663749 ou menores como 911382323 % M; ou aleatório [256..1e6).
- Módulos (M1, M2): 1_000_000_007 e 1_000_000_009 (primos).
- Alfabeto: mapear char -> int (>0). Ex: c = s[i]-'a'+1 para 'a'..'z'.
Operações úteis:
- build_hash(s): prefixos e potências de B
- get(l,r): hash normalizado da substring [l..r]
- concat(hA, lenA, hB): hash(A+B) a partir de hashes de A e B
- LCP via binária usando get(l,r)
*/

#include <bits/stdc++.h>
using namespace std;
using ull = unsigned long long;
using ll = long long;
struct DoubleHash {
static const ll M1 = 1000000007LL;
static const ll M2 = 1000000009LL;
```

```cpp
ll B;
vector<ll> p1, p2;
vector<ll> invp1, invp2;
vector<ll> h1, h2;

static ll modpow(ll a, ll e, ll m){
  ll r=1% m; a%=m;
  while(e){ if(e&1) r=(__int128)r*a % m; a=(__int128)a*a % m; e>>=1; }
  return r;
  static ll inv(ll a, ll m){
    return modpow(a, m-2, m);
  }
}

DoubleHash(const string& s, ll base = 0){
  mt19937_64 rng(chrono::high_resolution_clock::now().time_since_epoch().count());
  B = base ? base : (ll)(uniform_int_distribution<ll>(256, 1000000)(rng));
  int n = (int)s.size();
  p1.assign(n+1, 1); p2.assign(n+1, 1);
  invp1.assign(n+1, 1); invp2.assign(n+1, 1);
  h1.assign(n, 0); h2.assign(n, 0);
  ll invB1 = inv(B % M1, M1);
  ll invB2 = inv(B % M2, M2);

  for(int i=1;i<=n;i++){
    p1[i] = ( (__int128)p1[i-1] * (B % M1) ) % M1;
    p2[i] = ( (__int128)p2[i-1] * (B % M2) ) % M2;
    invp1[i] = ( (__int128)invp1[i-1] * invB1 ) % M1;
    invp2[i] = ( (__int128)invp2[i-1] * invB2 ) % M2;
  }

  auto val = [&](char c)->int{
    if('a'<=c && c<='z') return (c-'a'+1);
    if('A'<=c && c<='Z') return (c-'A'+27);
    if('0'<=c && c<='9') return (c-'0'+53);
    return 100 + (unsigned char)c;
  };

  if(n){
    h1[0] = val(s[0]) % M1;
    h2[0] = val(s[0]) % M2;
    for(int i=1;i<n;i++){
      h1[i] = ( ( (__int128)h1[i-1] + (__int128)val(s[i]) * p1[i] ) ) % M1;
      h2[i] = ( ( (__int128)h2[i-1] + (__int128)val(s[i]) * p2[i] ) ) % M2;
    }
  }
}

pair<ll,ll> get(int l, int r) const {
  if(l>r) return {0,0};
  ll x1 = h1[r];
  if(l) x1 = (x1 - h1[l-1] + M1) % M1;
  x1 = (__int128)x1 * invp1[l] % M1;
  ll x2 = h2[r];
  if(l) x2 = (x2 - h2[l-1] + M2) % M2;
  x2 = (__int128)x2 * invp2[l] % M2;
  return {x1, x2};
}

static pair<ll,ll> concat(const pair<ll,ll>& A, int lenA, const pair<ll,ll>& Bhash, int lenB,
ll B, ll M1=1000000007LL, ll M2=1000000009LL){
  auto modpow_ll = [&](ll a, int e, ll m){
    ll r=1% m; a%=m;
    while(e){ if(e&1) r=(__int128)r*a % m; a=(__int128)a*a % m; e>>=1; }
    return r;
  };

  ll pA1 = modpow_ll(B % M1, lenA, M1);
  ll pA2 = modpow_ll(B % M2, lenA, M2);
  ll c1 = ( ( (__int128)A.first + (__int128)Bhash.first * pA1 ) ) % M1;
  ll c2 = ( ( (__int128)A.second + (__int128)Bhash.second * pA2 ) ) % M2;
  return {c1, c2};
  }
};
```

Strings/kmp.cpp

Caminho: Strings/kmp.cpp

```
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
typedef vector<ll> v64;
#define forn(i, s, e) for (ll i = (s); i < (e); i++)

v64 prefix_function(string s) {
    ll n = s.size();
    v64 pi(n);
    forn (i, 1, n) {
        ll j = pi[i - 1];
        while (j > 0 && s[j] != s[i]) j = pi[j - 1];
        if (s[j] == s[i]) pi[i] = j + 1;
    }
    return pi;
}

v64 kmp(string s, string t) {
    ll n = s.size(), m = t.size();
    string st = s + '#' + t;
    v64 pi = prefix_function(st);
    v64 pos;
    forn (i, 0, pi.size()) {
        if (pi[i] == n) pos.push_back(i - 2 * n);
    }
    return pos;
}

v64 prefix_count(v64 pi) {
    ll n = pi.size();
    v64 ans(n + 1);
    for (ll i = 0; i < n; i++) ans[pi[i]]++;
    for (ll i = n - 1; i > 0; i--) ans[pi[i - 1]] += ans[i];
    for (ll i = 0; i <= n; i++) ans[i]++;
    return ans;
}

void compute_automaton(string s, vector<v64>& aut) {
    s += '#';
    ll n = s.size();
    v64 pi = prefix_function(s);
    aut.assign(n, v64(26));
    forn (i, 0, n) {
        for (char c = 'a'; c <= 'z'; c++) {
            if (i > 0 && c != s[i]) aut[i][c] = aut[pi[i - 1]][c];
            else aut[i][c] = i + (c == s[i]);
        }
    }
}
```