The 2025 ICPC Vietnam National Contest

HSGS Luong Van Tuy

LVTNB.Raumanian67

Giang Truong Vu, Thai Son Huu Vu, Nhat Minh Doan

09 November 2025

Contents

Contents

1 Data Structures

1.1 Fenwick Tree (BIT)

```cpp
struct Fenwick {
    int n;
    vector<ll> fen;

    Fenwick(int _n) : n(_n), fen(n + 1, 0) {}

    void modify(int p, ll val) {
        for (; p <= n; p += p & -p) fen[p] += val;
    }

    ll query(int p, ll res = 0) {
        for (; p > 0; p -= p & -p) res += fen[p];
        return res;
    }
};
```

1.2 Sparse Table

```cpp
struct SparseTable {
    static constexpr int lg = 20;
    int n;
    vector<vector<int>> st;
    vector<int> log;

    SparseTable(const vector<int> &arr) : n(arr.size()) {
        log.assign(n + 1, 0);
        for (int i = 2; i <= n; ++i) log[i] = log[i >> 1] + 1;

        st.assign(n, vector<int>(lg, 0));
        for (int i = 0; i < n; ++i) st[i][0] = arr[i];

        for (int j = 1; j < lg; ++j) {
            for (int i = 0; i + (1 << j) <= n; ++i) {
                st[i][j] = min(st[i][j - 1], st[i + (1 << (j - 1))][j - 1]);
            }
        }
    }

    int query(int l, int r) const {
        int j = log[r - l + 1];
        return min(st[l][j], st[r - (1 << j) + 1][j]);
    }
};
```

1.3 Disjoint Set Union (DSU)

```cpp
struct DSU {
    vector<int> parent, rank;

    void init(int n) {
        rank.assign(n + 1, 0);
        parent.resize(n + 1);
        iota(parent.begin(), parent.end(), 0);
    }

    int find(int u) { return u == parent[u] ? u : parent[u] = find(parent[u]); }

    void unite(int u, int v) {
        u = find(u), v = find(v);

        if (u == v) return;
        if (rank[u] < rank[v]) swap(u, v);
        if (rank[u] == rank[v]) ++rank[u];

        parent[v] = u;
    }
};
```

1.4 Lazy Segment Tree

```cpp
struct LazySegment {
    int n;
    vector<ll> tree, lazy;

    LazySegment(const vector<ll> &arr) : n(arr.size()) {
        tree.assign(n << 2, 0), lazy.assign(n << 2, 0);
        build(1, 0, n - 1, arr);
    }

    void modify(int p, ll val) { modify(1, 0, n - 1, p, val); }
    void range_modify(int l, int r, ll val) {
        range_modify(1, 0, n - 1, l, r, val);
    }
    ll query(int l, int r) { return query(1, 0, n - 1, l, r); }

private:
    void build(int node, int l, int r, const vector<ll> &arr) {
        if (l == r) return tree[node] = arr[l], void();

        int mid = l + r >> 1;

        build(node << 1, l, mid, arr);
        build(node << 1 | 1, -~mid, r, arr);
        tree[node] = tree[node << 1] + tree[node << 1 | 1];
    }

    void push(int node, int l, int r) {
        ll &val = lazy[node];

        if (!val) return;
        if (l ^ r) lazy[node << 1] += val, lazy[node << 1 | 1] += val;

        tree[node] += val * (r - l + 1);
        val = 0;
    }

    void range_modify(int node, int l, int r, int ql, int qr, ll val) {
        push(node, l, r);

        if (qr < l || r < ql) return;
        if (ql <= l && r <= qr) {
            lazy[node] += val;
            return push(node, l, r), void();
        }

        int mid = l + r >> 1;

        range_modify(node << 1, l, mid, ql, qr, val);
        range_modify(node << 1 | 1, -~mid, r, ql, qr, val);
        tree[node] = tree[node << 1] + tree[node << 1 | 1];
    }

    void modify(int node, int l, int r, int p, ll val) {
        if (l == r) return tree[node] = val, void();

        int mid = l + r >> 1;

        (p <= mid) ? modify(node << 1, l, mid, p, val)
                   : modify(node << 1 | 1, -~mid, r, p, val);
        tree[node] = tree[node << 1] + tree[node << 1 | 1];
    }

    ll query(int node, int l, int r, int ql, int qr) {
        push(node, l, r);

        if (qr < l || r < ql) return 0;
        if (ql <= l && r <= qr) return tree[node];

        int mid = l + r >> 1;

        return query(node << 1, l, mid, ql, qr) +
               query(node << 1 | 1, -~mid, r, ql, qr);
    }
};
```

1.5 Trie

```
struct Trie {
    struct Node {
        array<Node*, 26> child;
        bool isEnd;
        Node() : child{}, isEnd(false) {}
    };

    Node* root;

    Trie() : root(new Node()) {}

    void insert(const string &s) {
        Node* cur = root;
        for (char c : s) {
            int idx = c - 'a';
            if (!cur->child[idx]) cur->child[idx] = new Node();
            cur = cur->child[idx];
        }
        cur->isEnd = true;
    }

    bool search(const string &s) {
        Node* cur = root;
        for (char c : s) {
            int idx = c - 'a';
            if (!cur->child[idx]) return false;
            cur = cur->child[idx];
        }
        return cur->isEnd;
    }
};
```

2 Graph Theory

2.1 DFS & BFS

```cpp
vector<int> adj[N];
bool vis[N];

void dfs(int u) {
    vis[u] = true;
    for (int v : adj[u]) if (!vis[v]) dfs(v);
}

void bfs(int start) {
    queue<int> q;
    q.push(start), vis[start] = true;

    while (!q.empty()) {
        int u = q.front(); q.pop();
        for (int v : adj[u]) if (!vis[v]) {
            vis[v] = true, q.push(v);
        }
    }
}
```

2.2 Dijkstra's Algorithm

```cpp
vector<pair<int,int>> adj[N];
ll dist[N];

void dijkstra(int start, int n) {
    fill(dist, dist + n + 1, INF);
    priority_queue<pair<ll,int>, vector<pair<ll,int>>,
                   greater<pair<ll,int>>> pq;

    dist[start] = 0, pq.push({0, start});

    while (!pq.empty()) {
        auto [d, u] = pq.top(); pq.pop();

        if (d > dist[u]) continue;

        for (auto [v, w] : adj[u]) {
            if (dist[u] + w < dist[v]) {
                dist[v] = dist[u] + w;
                pq.push({dist[v], v});
            }
        }
    }
}
```

2.3 Bellman-Ford Algorithm

```cpp
struct Edge { int u, v; ll w; };
vector<Edge> edges;
ll dist[N];

bool bellman_ford(int start, int n) {
    fill(dist, dist + n + 1, INF);
    dist[start] = 0;

    for (int i = 0; i < n - 1; ++i) {
        for (auto [u, v, w] : edges) {
            if (dist[u] != INF && dist[u] + w < dist[v]) dist[v] = dist[u] + w;
        }
    }

    for (auto [u, v, w] : edges) {
        if (dist[u] != INF && dist[u] + w < dist[v]) return false;
    }
    return true;
}
```

2.4 Floyd-Warshall Algorithm

```
ll dist[N][N];

void floyd_warshall(int n) {
    for (int k = 1; k <= n; ++k) {
        for (int i = 1; i <= n; ++i) {
            for (int j = 1; j <= n; ++j) {
                dist[i][j] = min(dist[i][j], dist[i][k] + dist[k][j]);
            }
        }
    }
}
```

2.5 Kruskal's Algorithm

```
struct Edge {
    int u, v; ll w;
    bool operator<(const Edge &o) const { return w < o.w; }
};

vector<Edge> edges;
DSU dsu;

ll kruskal(int n) {
    sort(edges.begin(), edges.end());
    dsu.init(n);

    ll cost = 0;
    int cnt = 0;

    for (auto [u, v, w] : edges) {
        if (dsu.find(u) != dsu.find(v)) {
            dsu.unite(u, v);
            cost += w;
            if (++cnt == n - 1) break;
        }
    }
    return cost;
}
```

2.6 Tarjan's SCC Algorithm

```
vector<int> adj[N];
int low[N], disc[N], timer = 0;
bool onStack[N];
stack<int> st;
vector<vector<int>> sccs;

void tarjan(int u) {
    disc[u] = low[u] = ++timer;
    st.push(u), onStack[u] = true;

    for (int v : adj[u]) {
        if (!disc[v]) {
            tarjan(v);
            low[u] = min(low[u], low[v]);
        } else if (onStack[v]) low[u] = min(low[u], disc[v]);
    }

    if (low[u] == disc[u]) {
        vector<int> scc;
        while (true) {
            int v = st.top(); st.pop();
            onStack[v] = false;
            scc.push_back(v);
            if (v == u) break;
        }
        sccs.push_back(scc);
    }
}
```

2.7 Bridges & Articulation Points

```cpp
vector<int> adj[N];
int low[N], disc[N], timer = 0;
bool vis[N], is_art[N];
vector<pair<int,int>> bridges;

void dfs_bridge(int u, int p = -1) {
    vis[u] = true;
    disc[u] = low[u] = ++timer;
    int children = 0;

    for (int v : adj[u]) {
        if (v == p) continue;
        if (vis[v]) {
            low[u] = min(low[u], disc[v]);
        } else {
            dfs_bridge(v, u);
            low[u] = min(low[u], low[v]);

            if (low[v] > disc[u]) bridges.push_back({u, v});
            if (low[v] >= disc[u] && p != -1) is_art[u] = true;

            ++children;
        }
    }

    if (p == -1 && children > 1) is_art[u] = true;
}
```

2.8 Topological Sort

```cpp
vector<int> adj[N];
bool vis[N];
vector<int> topo;

void dfs_topo(int u) {
    vis[u] = true;
    for (int v : adj[u]) if (!vis[v]) dfs_topo(v);
    topo.push_back(u);
}

void topological_sort(int n) {
    for (int i = 1; i <= n; ++i) if (!vis[i]) dfs_topo(i);
    reverse(topo.begin(), topo.end());
}
```

2.9 Lowest Common Ancestor (LCA)

```cpp
constexpr int lg = 19;
vector<int> adj[N];
int depth[N], par[N][lg];

void dfs_lca(int u, int p) {
    par[u][0] = p;
    for (int j = 1; j < lg; ++j) par[u][j] = par[par[u][j - 1]][j - 1];
    for (int v : adj[u]) if (v != p) {
        depth[v] = depth[u] + 1;
        dfs_lca(v, u);
    }
}

int lca(int u, int v) {
    if (depth[u] < depth[v]) swap(u, v);
    for (int j = lg - 1; j >= 0; --j) {
        if (depth[par[u][j]] >= depth[v]) u = par[u][j];
    }
    for (int j = lg - 1; j >= 0; --j) {
        if (par[u][j] != par[v][j]) u = par[u][j], v = par[v][j];
    }
    return u == v ? u : par[u][0];
}
```

3 Dynamic Programming

3.1 Classic DP Patterns

```cpp
// 0/1 Knapsack
ll dp[N][N];

ll knapsack(vector<int> &wt, vector<int> &val, int W) {
    int n = wt.size();
    memset(dp, 0, sizeof dp);

    for (int i = 1; i <= n; ++i) {
        for (int w = 0; w <= W; ++w) {
            dp[i][w] = dp[i - 1][w];
            if (wt[i - 1] <= w) {
                dp[i][w] = max(dp[i][w], val[i - 1] + dp[i - 1][w - wt[i - 1]]);
            }
        }
    }
    return dp[n][W];
}

// Longest Increasing Subsequence (O(n log n))
int lis(vector<int> &arr) {
    vector<int> dp;
    for (int x : arr) {
        auto it = lower_bound(dp.begin(), dp.end(), x);
        if (it == dp.end()) dp.push_back(x);
        else *it = x;
    }
    return dp.size();
}

// Longest Common Subsequence
int lcs(string &s1, string &s2) {
    int n = s1.size(), m = s2.size();
    vector<vector<int>> dp(n + 1, vector<int>(m + 1, 0));

    for (int i = 1; i <= n; ++i) {
        for (int j = 1; j <= m; ++j) {
            dp[i][j] = (s1[i - 1] == s2[j - 1])
                     ? dp[i - 1][j - 1] + 1
                     : max(dp[i - 1][j], dp[i][j - 1]);
        }
    }
    return dp[n][m];
}
```

3.2 Digit DP

```cpp
int K;
ll dp[20][200][2];

ll digit_dp(const string &num, int pos, int sum, int tight) {
    if (pos == num.size()) return sum == K;
    if (dp[pos][sum][tight] != -1) return dp[pos][sum][tight];

    int limit = tight ? (num[pos] - '0') : 9;
    ll res = 0;

    for (int d = 0; d <= limit; ++d) {
        res += digit_dp(num, pos + 1, sum + d, tight && (d == limit));
    }

    return dp[pos][sum][tight] = res;
}
```

3.3 Bitmask DP

```
int n, dist[20][20];
ll dp[1 << 20][20];

ll tsp(int mask, int pos) {
    if (mask == (1 << n) - 1) return dist[pos][0];
    if (dp[mask][pos] != -1) return dp[mask][pos];

    ll ans = INF;
    for (int i = 0; i < n; ++i) if (!(mask & (1 << i))) {
        ans = min(ans, dist[pos][i] + tsp(mask | (1 << i), i));
    }
    return dp[mask][pos] = ans;
}
```

3.4 Tree DP

```
vector<int> adj[N];
ll dp[N][2];

void dfs_tree_dp(int u, int p) {
    dp[u][0] = dp[u][1] = 0;
    for (int v : adj[u]) if (v != p) {
        dfs_tree_dp(v, u);
        dp[u][0] += max(dp[v][0], dp[v][1]);
        dp[u][1] += dp[v][0];
    }
    dp[u][1] += 1;
}
```

4 Mathematics & Number Theory

4.1 Modular Arithmetic

```cpp
constexpr ll MOD = 1e9 + 7;

ll mod_add(ll a, ll b) { return ((a % MOD) + (b % MOD)) % MOD; }
ll mod_mul(ll a, ll b) { return ((a % MOD) * (b % MOD)) % MOD; }

ll mod_pow(ll base, ll exp) {
    ll res = 1;
    while (exp) {
        if (exp & 1) res = mod_mul(res, base);
        base = mod_mul(base, base);
        exp >>= 1;
    }
    return res;
}

ll mod_inv(ll a) { return mod_pow(a, MOD - 2); }
```

4.2 Modular Inverse (General)

```cpp
ll extgcd(ll a, ll b, ll &x, ll &y) {
    if (!b) return x = (a >= 0 ? 1 : -1), y = 0, llabs(a);
    ll g = extgcd(b, a % b, y, x);
    return y -= a / b * x, g;
}

ll modinv(ll a, ll m) {
    a = ((a % m) + m) % m;
    ll x, y, g = extgcd(a, m, x, y);
    return g != 1 ? -1 : ((x % m) + m) % m;
}
```

4.3 Prime Numbers

```cpp
vector<bool> is_prime;
vector<int> primes;

void sieve(int n) {
    is_prime.assign(n + 1, true);
    is_prime[0] = is_prime[1] = false;

    for (int i = 2; i <= n; ++i) if (is_prime[i]) {
        primes.push_back(i);
        for (ll j = (ll)i * i; j <= n; j += i) is_prime[j] = false;
    }
}

vector<pair<int,int>> factorize(int n) {
    vector<pair<int,int>> factors;
    for (int p : primes) {
        if (p * p > n) break;
        if (n % p == 0) {
            int cnt = 0;
            while (n % p == 0) n /= p, ++cnt;
            factors.push_back({p, cnt});
        }
    }
    if (n > 1) factors.push_back({n, 1});
    return factors;
}
```

4.4 Combinatorics

```cpp
ll fact[N], inv_fact[N];

void precompute(int n) {
    fact[0] = 1;
    for (int i = 1; i <= n; ++i) fact[i] = mod_mul(fact[i - 1], i);

    inv_fact[n] = mod_inv(fact[n]);
    for (int i = n - 1; i >= 0; --i) {
        inv_fact[i] = mod_mul(inv_fact[i + 1], i + 1);
    }
}

ll nCr(int n, int r) {
    if (r < 0 || r > n) return 0;
    return mod_mul(fact[n], mod_mul(inv_fact[r], inv_fact[n - r]));
}

ll catalan(int n) { return mod_mul(nCr(2 * n, n), mod_inv(n + 1)); }
```

4.5 Matrix Exponentiation

```cpp
struct Matrix {
    vector<vector<ll>> mat;
    int n;

    Matrix(int _n) : n(_n), mat(n, vector<ll>(n, 0)) {}

    Matrix operator*(const Matrix &o) const {
        Matrix res(n);
        for (int i = 0; i < n; ++i) {
            for (int j = 0; j < n; ++j) {
                for (int k = 0; k < n; ++k) {
                    res.mat[i][j] = mod_add(res.mat[i][j],
                                    mod_mul(mat[i][k], o.mat[k][j]));
                }
            }
        }
        return res;
    }
};

Matrix mat_pow(Matrix base, ll exp) {
    Matrix res(base.n);
    for (int i = 0; i < base.n; ++i) res.mat[i][i] = 1;

    while (exp) {
        if (exp & 1) res = res * base;
        base = base * base;
        exp >>= 1;
    }
    return res;
}

ll fib(ll n) {
    if (n <= 1) return n;
    Matrix base(2);
    base.mat = {{1, 1}, {1, 0}};
    return mat_pow(base, n - 1).mat[0][0];
}
```

5 String Algorithms

5.1 KMP Algorithm

```cpp
vector<int> kmp(const string &s) {
    vector<int> pi(s.size(), 0);

    for (int i = 1, j = 0; i < s.size(); ++i) {
        while (j && s[i] != s[j]) j = pi[j - 1];
        if (s[i] == s[j]) ++j;
        pi[i] = j;
    }

    return pi;
}

vector<int> kmp_search(const string &text, const string &pattern) {
    vector<int> pi = kmp(pattern);
    vector<int> matches;

    for (int i = 0, j = 0; i < text.size(); ++i) {
        while (j && text[i] != pattern[j]) j = pi[j - 1];
        if (text[i] == pattern[j]) ++j;
        if (j == pattern.size()) {
            matches.push_back(i - j + 1);
            j = pi[j - 1];
        }
    }
    return matches;
}
```

5.2 Z-Algorithm

```cpp
vector<int> z_algorithm(const string &s) {
    int n = s.size();
    vector<int> z(n, 0);
    int l = 0, r = 0;

    for (int i = 1; i < n; ++i) {
        if (i <= r) z[i] = min(r - i + 1, z[i - l]);
        while (i + z[i] < n && s[z[i]] == s[i + z[i]]) ++z[i];
        if (i + z[i] - 1 > r) l = i, r = i + z[i] - 1;
    }
    return z;
}

vector<int> z_search(const string &text, const string &pattern) {
    string combined = pattern + "$" + text;
    vector<int> z = z_algorithm(combined);
    vector<int> matches;
    int m = pattern.size();

    for (int i = m + 1; i < combined.size(); ++i) {
        if (z[i] == m) matches.push_back(i - m - 1);
    }
    return matches;
}
```

5.3 String Hashing

```cpp
struct Rabinkarp {
    static constexpr int mod = 1e9 + 7, base = 256;

    int n;
    vector<ll> hash, power;

    Rabinkarp(const string &s) : n(s.size()), hash(n + 1, 0),
                                 power(n + 1, 1) {
        for (int i = 0; i < n; ++i) {
            power[i + 1] = power[i] * base % mod;
            hash[i + 1] = (hash[i] * base + (unsigned char)s[i]) % mod;
        }
    }

    ll get(int l, int r) const {
        return ((hash[r + 1] - hash[l] * power[r - l + 1]) % mod + mod) % mod;
    }
};
```

5.4 Manacher's Algorithm

```cpp
string preprocess(const string &s) {
    string t = "@";
    for (char c : s) t += "#", t += c;
    return t += "#$";
}

vector<int> manacher(const string &s) {
    string t = preprocess(s);
    int n = t.size();
    vector<int> p(n, 0);
    int c = 0, r = 0;

    for (int i = 1; i < n - 1; ++i) {
        int mirror = 2 * c - i;
        if (i < r) p[i] = min(r - i, p[mirror]);

        while (t[i + p[i] + 1] == t[i - p[i] - 1]) ++p[i];

        if (i + p[i] > r) c = i, r = i + p[i];
    }
    return p;
}
```

6 Miscellaneous

6.1 Mo's Algorithm

```cpp
constexpr int BLOCK = 320;

struct Query {
    int l, r, idx;
    bool operator<(const Query &o) const {
        int bl = l / BLOCK, br = o.l / BLOCK;
        if (bl != br) return bl < br;
        return (bl & 1) ? (r < o.r) : (r > o.r);
    }
};

int cur_ans = 0;

void add(int pos) { /* update cur_ans */ }
void remove(int pos) { /* update cur_ans */ }

void mo_algorithm(vector<Query> &queries) {
    sort(queries.begin(), queries.end());
    vector<int> ans(queries.size());

    int l = 0, r = -1;
    for (auto q : queries) {
        while (l > q.l) add(--l);
        while (r < q.r) add(++r);
        while (l < q.l) remove(l++);
        while (r > q.r) remove(r--);
        ans[q.idx] = cur_ans;
    }
}
```

6.2 Meet in the Middle

```cpp
bool subset_sum_mitm(const vector<int> &arr, int target) {
    int n = arr.size(), mid = n >> 1;

    unordered_set<int> first;
    for (int mask = 0; mask < (1 << mid); ++mask) {
        int sum = 0;
        for (int i = 0; i < mid; ++i) {
            if (mask & (1 << i)) sum += arr[i];
        }
        first.insert(sum);
    }

    for (int mask = 0; mask < (1 << (n - mid)); ++mask) {
        int sum = 0;
        for (int i = 0; i < n - mid; ++i) {
            if (mask & (1 << i)) sum += arr[mid + i];
        }
        if (first.count(target - sum)) return true;
    }
    return false;
}
```