



Notebook - Competitive Programming

Anões do TLE

Contents					
1 Data structures	2	4.3 Graph	3	6 Problems	5
1.1 Ufds	2	4.4 Retrieve Path	4	6.1 Kth Digit String (CSES)	5
2 Dynamic programming	2	5 Math	4	7 Strings	6
2.1 Kadane	2	5.1 Binomial	4	7.1 Manacher	6
2.2 Longest Increasing Subsequence (LIS)	2	5.2 Count Divisors	4	8 Trees	6
3 Geometry	2	5.3 Factorization With Sieve	4	8.1 LCA Binary Lifting (CP Algo)	6
3.1 Convex Hull	2	5.4 Factorization	4	8.2 LCA SegTree (CP Algo)	6
3.2 Point To Segment	3	5.5 Fast Exp Iterative	4	8.3 LCA Sparse Table	7
4 Graphs	3	5.6 Fast Exp	4	8.4 Tree Isomorph	8
4.1 Dijkstra Path	3	5.7 GCD	5	9 Settings and macros	8
4.2 Dijkstra	3	5.8 Is prime	5	9.1 macro.cpp	8
		5.9 LCM	5	9.2 short-macro.cpp	8
		5.10 Euler phi $\varphi(n)$	5		
		5.11 Sieve	5		
		5.12 Sum Divisors	5		

1 Data structures

1.1 Ufds

```
class UFDS {
private:
    vector<int> size, ps;

public:
    UFDS(int N) : size(N + 1, 1), ps(N + 1) { iota(ps.begin(), ps.end(), 0); }

    int find_set(int x) { return x == ps[x] ? x : (ps[x] = find_set(ps[x])); }

    bool same_set(int x, int y) { return find_set(x) == find_set(y); }

    void union_set(int x, int y) {
        if (same_set(x, y)) return;

        int p = find_set(x);
        int q = find_set(y);

        if (size[p] < size[q]) swap(p, q);

        ps[q] = p;
        size[p] += size[q];
    }
};
```

2 Dynamic programming

2.1 Kadane

```
int kadane(const vi& xs) {
    vi s(xs.size());
    s[0] = xs[0];

    for (size_t i = 1; i < xs.size(); ++i) s[i] = max(xs[i], s[i - 1] + xs[i]);

    return *max_element(all(s));
}
```

2.2 Longest Increasing Subsequence (LIS)

Time: $O(N \cdot \log N)$.

```
int lis(vi const& a) {
    int n = a.size();
    const int INF = 1e9;
    vi d(n + 1, INF);
    d[0] = -INF;

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

    int ans = 0;
```

```
    for (int l = 0; l <= n; l++) {
        if (d[l] < INF) ans = l;
    }

    return ans;
}
```

3 Geometry

3.1 Convex Hull

Given a set of points find the smallest convex polygon that contains all the given points.

Time: $O(N \cdot \log N)$

By default it removes the collinear points, set the boolean to true if you don't want that

```
struct pt {
    double x, y;
};

int orientation(pt a, pt b, pt c) {
    double v = a.x * (b.y - c.y) + b.x * (c.y - a.y) + c.x * (a.y - b.y);
    if (v < 0) return -1; // clockwise
    if (v > 0) return +1; // counter-clockwise
    return 0;
}

bool cw(pt a, pt b, pt c, bool include_collinear) {
    int o = orientation(a, b, c);
    return o < 0 || (include_collinear && o == 0);
}

bool collinear(pt a, pt b, pt c) { return orientation(a, b, c) == 0; }

void convex_hull(vector<pt>& a, bool include_collinear = false) {
    pt p0 = *min_element(a.begin(), a.end(), [](pt a, pt b) {
        return make_pair(a.y, a.x) < make_pair(b.y, b.x);
    });
    sort(a.begin(), a.end(), [&p0](const pt& a, const pt& b) {
        int o = orientation(p0, a, b);
        if (o == 0)
            return (p0.x - a.x) * (p0.x - a.x) + (p0.y - a.y) * (p0.y - a.y) <
                (p0.x - b.x) * (p0.x - b.x) + (p0.y - b.y) * (p0.y - b.y);
        return o < 0;
    });
    if (include_collinear) {
        int i = (int)a.size() - 1;
        while (i >= 0 && collinear(p0, a[i], a.back())) i--;
        reverse(a.begin() + i + 1, a.end());
    }

    vector<pt> st;
    for (int i = 0; i < (int)a.size(); i++) {
        while (st.size() > 1 &&
            !cw(st[st.size() - 2], st.back(), a[i], include_collinear))
            st.pop_back();
        st.push_back(a[i]);
    }
}
```

```

    a = st;
}

```

3.2 Point To Segment

```

typedef pair<double, double> pdb;

```

```

#define fst first
#define snd second

```

```

double pt2segment(pdb A, pdb B, pdb E) {
    pdb AB = {B.fst - A.fst, B.snd - A.snd};
    pdb BE = {E.fst - B.fst, E.snd - B.snd};
    pdb AE = {E.fst - A.fst, E.snd - A.snd};

```

```

    double AB_BE = AB.fst * BE.fst + AB.snd * BE.snd;
    double AB_AE = AB.fst * AE.fst + AB.snd * AE.snd;

```

```

    double ans;
    if (AB_BE > 0) {
        double y = E.snd - B.snd;
        double x = E.fst - B.fst;
        ans = sqrt(x * x + y * y);
    } else if (AB_AE < 0) {
        double y = E.snd - A.snd;
        double x = E.fst - A.fst;
        ans = sqrt(x * x + y * y);
    } else {
        auto [x1, y1] = AB;
        auto [x2, y2] = AE;
        double mod = sqrt(x1 * x1 + y1 * y1);
        ans = abs(x1 * y2 - y1 * x2) / mod;
    }

```

```

    return ans;
}

```

4 Graphs

4.1 Dijkstra Path

```

pair<v1, v1> Graph::dijkstra_path(ll src) {
    v1 pd(this->N, LLONG_MAX), ds(this->N, LLONG_MAX);
    pd[src] = src;
    ds[src] = 0;

```

```

    set<pll> st;
    st.emplace(0, src);

```

```

    while (!st.empty()) {
        ll u = st.begin()->snd;
        ll wu = st.begin()->fst;
        st.erase(st.begin());

```

```

        if (wu != ds[u]) continue;
        for (auto& [v, w] : adj[u]) {

```

```

            if (ds[v] > ds[u] + w) {
                ds[v] = ds[u] + w;
                pd[v] = u;
                st.emplace(ds[v], v);
            }
        }
    }

    return {ds, pd};
}

```

4.2 Dijkstra

```

v1 Graph::dijkstra(ll src) {
    v1 ds(this->N, LLONG_MAX);
    ds[src] = 0;

```

```

    set<pll> st;
    st.emplace(0, src);

```

```

    while (!st.empty()) {
        ll u = st.begin()->snd;
        ll wu = st.begin()->fst;
        st.erase(st.begin());

```

```

        if (wu != ds[u]) continue;
        for (auto& [v, w] : adj[u]) {
            if (ds[v] > ds[u] + w) {
                ds[v] = ds[u] + w;
                st.emplace(ds[v], v);
            }
        }
    }

```

```

    return ds;
}

```

4.3 Graph

```

class Graph {
private:
    ll N;
    bool undirected;
    vector<v1ll> adj;

```

```

public:
    Graph(ll N, bool is_undirected = true) {
        this->N = N;
        adj.resize(N);
        undirected = is_undirected;
    }

```

```

    void add(ll u, ll v, ll w) {
        adj[u].emplace_back(v, w);
        if (undirected) adj[v].emplace_back(u, w);
    }

```

```

    v1 dijkstra(ll src);

```

```

    pair<vl, vl> dijkstra_path(ll src);

    vl retrieve_path(ll s, ll u, const vl& pred);
};

```

4.4 Retrieve Path

```

vl Graph::retrieve_path(ll src, ll trg, const vl& pred) {
    vl p;
    ll v = trg;

    do {
        p.emplace_back(pred[v], v);
        v = pred[v];
    } while (v != src);

    reverse(all(p));

    return p;
}

```

5 Math

5.1 Binomial

```

ll binom(ll n, ll k) {
    if (k > n) return 0;
    vl dp(k + 1, 0);
    dp[0] = 1;
    for (ll i = 1; i <= n; i++)
        for (ll j = k; j > 0; j--) dp[j] = dp[j] + dp[j - 1];
    return dp[k];
}

```

5.2 Count Divisors

```

ll count_divisors(ll num) {
    ll count = 1;
    for (int i = 2; (ll)i * i <= num; i++) {
        if (num % i == 0) {
            int e = 0;
            do {
                e++;
                num /= i;
            } while (num % i == 0);
            count *= e + 1;
        }
    }
    if (num > 1) {
        count *= 2;
    }
    return count;
}

```

5.3 Factorization With Sieve

```

map<ll, ll> factorization_with_sieve(ll n, const vl& primes) {
    map<ll, ll> fact;

    for (ll d : primes) {
        if (d * d > n) break;

        ll k = 0;
        while (n % d == 0) {
            k++;
            n /= d;
        }

        if (k) fact[d] = k;
    }

    if (n > 1) fact[n] = 1;
    return fact;
}

```

5.4 Factorization

```

map<ll, ll> factorization(ll n) {
    map<ll, ll> ans;
    for (ll i = 2; i * i <= n; i++) {
        ll count = 0;
        for (; n % i == 0; count++, n /= i)
            ;
        if (count) ans[i] = count;
    }
    if (n > 1) ans[n]++;
    return ans;
}

```

5.5 Fast Exp Iterative

```

long long fast_exp_it(long long a, int n) {
    long long res = 1, base = a;

    while (n) {
        if (n & 1) res *= base;

        base *= base;
        n >>= 1;
    }

    return res;
}

```

5.6 Fast Exp

```

long long fast_exp(long long a, int n) {
    if (n == 1) return a;

    auto x = fast_exp(a, n / 2);

    return x * x * (n % 2 ? a : 1);
}

```

5.7 GCD

The Euclidean algorithm allows to find the greatest common divisor of two numbers a and b in $O(\log \cdot \min(a, b))$.

```
11 gcd(11 a, 11 b) { return b ? gcd(b, a % b) : a; }
```

5.8 Is prime

$O(\sqrt{N})$

```
bool isprime(11 n) {
    if (n < 2) return false;
    if (n == 2) return true;
    if (n % 2 == 0) return false;
    for (11 i = 3; i * i < n; i += 2)
        if (n % i == 0) return false;
    return true;
}
```

5.9 LCM

Calculating the least common multiple (commonly denoted LCM) can be reduced to calculating the GCD with the following simple formula: $\text{lcm}(a, b) = (a \cdot b) / \text{gcd}(a, b)$
Thus, LCM can be calculated using the Euclidean algorithm with the same time complexity:

```
11 lcm(11 a, 11 b) { return a / gcd(a, b) * b; }
```

5.10 Euler phi $\varphi(n)$

Computes the number of positive integers less than n that are coprimes with n , in $O(\sqrt{N})$.

```
11 phi(11 n) {
    if (n == 1) return 1;

    auto fs = factorization(n);
    auto res = n;

    for (auto [p, k] : fs) {
        res /= p;
        res *= (p - 1);
    }

    return res;
}
```

5.11 Sieve

```
v1 sieve(11 N) {
    bitset<MAX + 1> sieve;
    v1 ps{2, 3};
    sieve.set();

    for (11 i = 5, step = 2; i <= N; i += step, step = 6 - step) {
        if (sieve[i]) {
            ps.push_back(i);

```

```
        for (11 j = i * i; j <= N; j += 2 * i) sieve[j] = false;
        }
    }
    return ps;
}
```

5.12 Sum Divisors

```
11 sum_divisors(11 num) {
    11 result = 1;

    for (int i = 2; (11)i * i <= num; i++) {
        if (num % i == 0) {
            int e = 0;
            do {
                e++;
                num /= i;
            } while (num % i == 0);

            11 sum = 0, pow = 1;
            do {
                sum += pow;
                pow *= i;
            } while (e-- > 0);
            result *= sum;
        }
    }
    if (num > 1) {
        result *= (1 + num);
    }
    return result;
}
```

6 Problems

6.1 Kth Digit String (CSES)

Time: $O(\log_{10} K)$.

Space: $O(1)$.

```
11 kth_digit_string(11 k) {
    if (k < 10) return k;

    11 c = 180, i = 2, u = 10, r = 0, ans = -1, m;
    for (k -= 9; k > c; i++, u *= 10) {
        k -= c;
        c /= i;
        c *= 10 * (i + 1);
    }

    if ((m = k % i))
        r++;
    else
        m = i;

    11 tmp = (k / i) + r + u - 1;
    for (m = i + 1 - m; m--; tmp /= 10) ans = tmp % 10;
}
```

```
    return ans;
}
```

7 Strings

7.1 Manacher

Given string s with length n . Find all the pairs (i, j) such that substring $s[i \dots j]$ is a palindrome. String t is a palindrome when $t = t_{rev}$ (t_{rev} is a reversed string for t).

Time: $O(N)$

```
vi manacher(string s) {
    string t;
    for (auto c : s) t += string("#") + c;
    t = t + '?';

    int n = t.size();
    t = "$" + t + "^";

    vi p(n + 2);
    int l = 1, r = 1;
    for (int i = 1; i <= n; i++) {
        p[i] = max(0, min(r - i, p[l + (r - i)]));
        while (t[i - p[i]] == t[i + p[i]]) p[i]++;
        if (i + p[i] > r) {
            l = i - p[i], r = i + p[i];
        }
        p[i]--;
    }

    return vi(begin(p) + 1, end(p) - 1);
}
```

8 Trees

8.1 LCA Binary Lifting (CP Algo)

The algorithm described will need $O(N \cdot \log N)$ for preprocessing the tree, and then $O(\log N)$ for each LCA query.

```
ll n, l;
vector<ll> adj[MAX];

ll timer;
vector<ll> tin, tout;
vector<vector<ll>> up;

void dfs(ll v, ll p) {
    tin[v] = ++timer;
    up[v][0] = p;
    for (ll i = 1; i <= l; ++i) up[v][i] = up[up[v][i - 1]][i - 1];

    for (ll u : adj[v]) {
        if (u != p) dfs(u, v);
    }
}
```

```
tout[v] = ++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, vector<ll>(l + 1));
    dfs(root, root);
}
```

8.2 LCA SegTree (CP Algo)

The algorithm can answer each query in $O(\log N)$ with preprocessing in $O(N)$ time.

```
struct LCA {
    vector<ll> height, euler, first, segtree;
    vector<bool> visited;
    ll n;

    LCA(vector<vector<ll>>& adj, ll root = 0) {
        n = adj.size();
        height.resize(n);
        first.resize(n);
        euler.reserve(n * 2);
        visited.assign(n, false);
        dfs(adj, root);
        ll m = euler.size();
        segtree.resize(m * 4);
        build(1, 0, m - 1);
    }

    void dfs(vector<vector<ll>>& adj, ll node, ll h = 0) {
        visited[node] = true;
        height[node] = h;
        first[node] = euler.size();
        euler.push_back(node);
        for (auto to : adj[node]) {
            if (!visited[to]) {
                dfs(adj, to, h + 1);
                euler.push_back(node);
            }
        }
    }
}
```

```

void build(ll node, ll b, ll e) {
    if (b == e) {
        segtree[node] = euler[b];
    } else {
        ll mid = (b + e) / 2;
        build(node << 1, b, mid);
        build(node << 1 | 1, mid + 1, e);
        ll l = segtree[node << 1], r = segtree[node << 1 | 1];
        segtree[node] = (height[l] < height[r]) ? l : r;
    }
}

ll query(ll node, ll b, ll e, ll L, ll R) {
    if (b > R || e < L) return -1;
    if (b >= L && e <= R) return segtree[node];
    ll mid = (b + e) >> 1;

    ll left = query(node << 1, b, mid, L, R);
    ll right = query(node << 1 | 1, mid + 1, e, L, R);
    if (left == -1) return right;
    if (right == -1) return left;
    return height[left] < height[right] ? left : right;
}

ll lca(ll u, ll v) {
    ll left = first[u], right = first[v];
    if (left > right) swap(left, right);
    return query(1, 0, euler.size() - 1, left, right);
}
};

```

8.3 LCA Sparse Table

The algorithm described will need $O(N)$ for preprocessing, and then $O(1)$ for each LCA query.
0 indexed !

```

#define len(__x) (int)__x.size()

using ll = long long;
using pll = pair<ll, ll>;
using vi = vector<int>;
using vi2d = vector<vi>;
#define all(a) a.begin(), a.end()
#define pb(____x) push_back(____x)
#define mp(____a, ____b) make_pair(____a, ____b)
#define eb(____x) emplace_back(____x)

template <typename T>
struct SparseTable {
    vector<T> v;
    ll n;
    static const ll b = 30;
    vi mask, t;

    ll op(ll x, ll y) { return v[x] < v[y] ? x : y; }
    ll msb(ll x) { return __builtin_clz(1) - __builtin_clz(x); }
    SparseTable() {}

```

```

SparseTable(const vector<T>& v_) : v(v_), n(v.size()), mask(n), t(n) {
    for (ll i = 0, at = 0; i < n; mask[i++] = at |= 1) {
        at = (at << 1) & ((1 << b) - 1);
        while (at and op(i, i - msb(at & -at)) == i) at ^= at & -at;
    }
    for (ll i = 0; i < n / b; i++)
        t[i] = b * i + b - 1 - msb(mask[b * i + b - 1]);
    for (ll j = 1; (1 << j) <= n / b; j++)
        for (ll i = 0; i + (1 << j) <= n / b; i++)
            t[n / b * j + i] =
                op(t[n / b * (j - 1) + i], t[n / b * (j - 1) + i + (1 << (j - 1))]);
}

ll small(ll r, ll sz = b) { return r - msb(mask[r] & ((1 << sz) - 1)); }
T query(ll l, ll r) {
    if (r - l + 1 <= b) return small(r, r - l + 1);
    ll ans = op(small(l + b - 1), small(r));
    ll x = l / b + 1, y = r / b - 1;
    if (x <= y) {
        ll j = msb(y - x + 1);
        ans = op(ans, op(t[n / b * j + x], t[n / b * j + y - (1 << j) + 1]));
    }
    return ans;
}
};

struct LCA {
    SparseTable<ll> st;
    ll n;
    vi v, pos, dep;

    LCA(const vi2d& g, ll root) : n(len(g)), pos(n) {
        dfs(root, 0, -1, g);
        st = SparseTable<ll>(vector<ll>(all(dep)));
    }

    void dfs(ll i, ll d, ll p, const vi2d& g) {
        v.pb(len(dep)) = i, pos[i] = len(dep), dep.pb(d);
        for (auto j : g[i])
            if (j != p) {
                dfs(j, d + 1, i, g);
                v.pb(len(dep)) = i, dep.pb(d);
            }
    }

    ll lca(ll a, ll b) {
        ll l = min(pos[a], pos[b]);
        ll r = max(pos[a], pos[b]);
        return v[st.query(l, r)];
    }

    ll dist(ll a, ll b) {
        return dep[pos[a]] + dep[pos[b]] - 2 * dep[pos[lca(a, b)]];
    }
};

```

8.4 Tree Isomorph

Checks whether two tree are isomorph. The function thash() returns the hash of the tree (using centroids as special vertices). Two trees are isomorph if their hash are the same.

```
map<vector<int>, int> mhash;
```

```
struct tree {
    int n;
    vector<vector<int>> g;
    vector<int> sz, cs;

    tree(int n_) : n(n_), g(n_), sz(n_) {}

    void dfs_centroid(int v, int p) {
        sz[v] = 1;
        bool cent = true;
        for (int u : g[v])
            if (u != p) {
                dfs_centroid(u, v), sz[v] += sz[u];
                if (sz[u] > n / 2) cent = false;
            }
        if (cent and n - sz[v] <= n / 2) cs.push_back(v);
    }

    int fhash(int v, int p) {
        vector<int> h;
        for (int u : g[v])
            if (u != p) h.push_back(fhash(u, v));
        sort(h.begin(), h.end());
        if (!mhash.count(h)) mhash[h] = mhash.size();
        return mhash[h];
    }

    ll thash() {
        cs.clear();
        dfs_centroid(0, -1);
        if (cs.size() == 1) return fhash(cs[0], -1);
        ll h1 = fhash(cs[0], cs[1]), h2 = fhash(cs[1], cs[0]);
        return (min(h1, h2) << 30) + max(h1, h2);
    }

    void add(int a, int b) {
        g[a].emplace_back(b);
        g[b].emplace_back(a);
    }
};
```

9 Settings and macros

9.1 macro.cpp

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

```
using namespace std;

typedef long long ll;
typedef pair<int, int> pii;
typedef pair<ll, ll> pll;
typedef vector<int> vi;
typedef vector<ll> vl;
typedef vector<pii> vii;
typedef vector<pll> vll;

#define fst first
#define snd second
#define all(x) x.begin(), x.end()
#define vin(vt) for (auto &e : vt) cin >> e
#define LSOne(S) ((S) & -(S))
#define MSOne(S) (1ull << (63 - __builtin_clzll(S)))
#define fastio ios_base::sync_with_stdio(0); \
    cin.tie(0); \
    cout.tie(0)

const vii dir4 {{1,0},{-1,0},{0,1},{0,-1}};

auto solve() { }

int main() {
    fastio;

    ll t = 1;
    //cin >> t;

    while (t--) solve();

    return 0;
}
```

9.2 short-macro.cpp

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

using namespace std;

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

#define all(x) x.begin(), x.end()
#define vin(vt) for (auto &e : vt) cin >> e

auto solve() { }

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

    ll t = 1;
    //cin >> t;

    while (t--) solve();
}
```



```
return 0;
```

```
    }
```