Competitve Programming Note

youngyangze(Hanbyeol Lee)

Compiled on March 6, 2025

Contents

1 Math **2**
- 1.1 Exponentiation by Squaring . . . 2
- 1.2 Euler Phi . . . 2
- 1.3 Xudyh's sieve . . . 2
- 1.4 Berlekamp Massey . . . 2
- 1.5 FFT w/ NTT . . . 3
  - 1.5.1 FFT? . . . 3
- 1.6 Kitamasa . . . 3
- 1.7 Mod Int . . . 3
- 1.8 Miller Rabin w/ Pollard Rho . . . 3
- 1.9 Simplex . . . 4
- 1.10 De Brujin Sequence . . . 4

2 Geometry **4**
- 2.1 Convex Hull . . . 4
- 2.2 Dynamic Convex Hull . . . 4
- 2.3 3D Convex Hull . . . 4
- 2.4 Smallest Enclosing Sphere/Circle . . . 4
- 2.5 K-D Tree . . . 4
- 2.6 Half Plane Intersection . . . 4
- 2.7 Delauney w/ Vornoi Diagram . . . 4

3 Data Structure **4**
- 3.1 Segment Tree w/ Lazy . . . 4
- 3.2 Splay Tree . . . 4
- 3.3 Link Cut Tree . . . 4
- 3.4 Lichao Tree . . . 4

4 Strings **4**
- 4.1 Manacher . . . 4
- 4.2 Hirschberg . . . 4
- 4.3 Round LCS . . . 4
- 4.4 Z . . . 4
- 4.5 Suffix and LCP . . . 4
- 4.6 Palindrome Tree . . . 4

5 Graph **4**
- 5.1 Dinic . . . 4
- 5.2 SCC . . . 4
- 5.3 2-SAT . . . 4
- 5.4 Directed MST . . . 4
- 5.5 SCC . . . 4
- 5.6 Global Min Cut . . . 4
- 5.7 General Matching . . . 4
- 5.8 Centroid Decomposition . . . 4
- 5.9 Bipartite Matching . . . 4
- 5.10 Dominator Tree . . . 4
- 5.11 MCMF . . . 4
- 5.12 Gomory Hu Tree . . . 4
- 5.13 BCC . . . 4
- 5.14 Block Cut Tree . . . 4
- 5.15 Cut Vertex/Edge . . . 4

6 Misc. **5**
- 6.1 Basic Template . . . 5
- 6.2 Primes . . . 5

PLEASE CHECK THE STATEMENT AGAIN
EDITOR SHOULD ADD SOME COMMENTS IN EACH THINGS

1 Math

1.1 Exponentiation by Squaring

A fast exponentiation, works in $O(\log k)$

```
ll pow1(ll n, ll k) {
    ll ret = 1;
    while (k) {
        if (k & 1) ret *= n;
        n *= n;
        k >>= 1;
    }
    return ret;
}

ll pow2(ll n, ll k, ll mod) {
    if (mod == 1) return 0;
    ll ret = 1;
    n %= mod;
    while (k > 0) {
        if (k & 1) ret = (ret * n) % mod;
        k >>= 1;
        n = (n * n) % mod;
    }
    return ret;
}
```

1.2 Euler Phi

Calculates $\phi(n)$ in $O(\sqrt{n})$
Formula of $\phi(n)$: $|k \in \{1, \ldots, n\} : \gcd n, k = 1|$

```
ll phi(ll n) {
    ll ret = n;
    for (ll i = 2; i * i <= n; i++) {
        if (n % i == 0) {
            while (n % i == 0) n /= i;
            ret -= ret / i;
        }
    }
    if (1 < n) ret -= ret / n;
    return ret;
}
```

1.3 Xudyh's sieve

Finds $M(n)$(known as Mertens function) in $O(N^{\frac{2}{3}})$ where $M(n) = \sum_{i=1}^{n} \mu(i)$

```
const int MAX = 15000009;

bool chk[MAX + 1];
ll mu(MAX + 1, 1), pri, pre(MAX + 1, 0);

void init() {
    memset(chk, 1, sizeof(chk));
    memset(mu, 1, sizeof(mu));
    memset(pre, 0, sizeof(pre));
    mu[1] = 1;
    for (int i = 2; i <= MAX; i++) {
        if (chk[i]) {
            pri.emplace_back(i);
            mu[i] = -1;
        }
        for (int p : pri) {
            if ((ll)i * p > MAX) break;
            chk[i * p] = 0;
            if (i % p == 0) {
                mu[i * p] = 0;
                break;
            } else mu[i * p] = -mu[i];
        }
    }
    pre[0] = 0;
    for (int i = 1; i <= MAX; i++) pre[i] = pre[i - 1] + mu[i];
}

unordered_map<ll, ll> cache;
ll get(ll n) {
    if (n <= MAX) return pre[n];
    if (cache.count(n)) return cache[n];
    ll ret = 1, i = 2;
    while (i <= n) {
        ll j = n / (n / i);
        ret -= (j - i + 1) * get(n / i);
        i = j + 1;
    }
    return cache[n] = ret;
}
```

1.4 Berlekamp Massey

Requires pow2. Works in $O(N^2)$

```
const int MOD = 1e9 + 7;

vll berlekamp_massey(vll x) {
    vll ls, current;
    ll lf, ld;
    for (ll i = 0; i < x.size(); i++) {
        ll t = 0;
        for (ll j = 0; j < current.size(); j++) t = (t + 1LL * x[i - j - 1] * current[j]) %
        MOD;
        if ((t - x[i]) % MOD == 0) continue;
        if (current.empty()) {
            current.resize(i + 1);
            lf = i;
            ld = (t - x[i]) % MOD;
            continue;
        }
        ll k = -(x[i] - t) * _pow(ld, MOD - 2) % MOD;
        vll c(i - lf - 1);
        c.emplace_back(k);
        for (auto &j : ls) c.emplace_back(-j * k % MOD);
        if (c.size() < current.size()) c.resize(current.size());
        for (ll j = 0; j < current.size(); j++) c[j] = (c[j] + current[j]) % MOD;
```

```
        if (i - lf + (ll)ls.size() >= (ll)current.size()) tie(ls, lf, ld) =
        make_tuple(current, i, (t - x[i]) % MOD);
        current = c;
    }
    for (auto &i : current) i = (i % MOD + MOD) % MOD;
    return current;
}

ll get_nth(vll rec, vll dp, ll n) {
    ll m = rec.size();
    vll s(m), t(m);
    s[0] = 1;
    if (m != 1) t[1] = 1;
    else t[0] = rec[0];
    auto mult = [&rec](vll v, vll w) {
        ll m = v.size();
        vll t(2 * m);
        for (ll j = 0; j < m; j++) {
            for (ll k = 0; k < m; k++) {
                t[j + k] += 1LL * v[j] * w[k] % MOD;
                if (t[j + k] >= MOD) t[j + k] -= MOD;
            }
        }
        for (ll j = 2 * m - 1; j >= m; j--) {
            for (ll k = 1; k <= m; k++) {
                t[j - k] += 1LL * t[j] * rec[k - 1] % MOD;
                if (t[j - k] >= MOD) t[j - k] -= MOD;
            }
        }
        t.resize(m);
        return t;
    };
    while (n) {
        if (n & 1) s = mult(s, t);
        t = mult(t, t);
        n >>= 1;
    }
    ll ret = 0;
    for (ll i = 0; i < m; i++) ret += 1LL * s[i] * dp[i] % MOD;
    return ret % MOD;
}

ll guess(vll x, ll n) {
    if (n < x.size()) return x[n];
    vll v = berlekamp_massey(x);
    if (v.empty()) return 0;
    return get_nth(v, x, n);
}
```

1.5 FFT w/ NTT

Should be **added**.

1.5.1 FFT?

Should be **added**.

1.6 Kitamasa

WIP

```
constexpr int W = 3;
using mint = modint<MOD>;
using poly = mpoly<W, MOD>;

mint kitamasa(poly c, poly a, ll n) {
    poly d = vector<mint>{1}, xn = vector<mint>{0, 1}, f;
    for (int i = 0; i < c.a.size(); i++) f.push_back(-c.a[i]);
    f.push_back(1);
    while (n) {
        if (n & 1) d = d * xn % f;
        n >>= 1;
        xn = xn * xn % f;
    }

    d.a.resize(a.size(), 0);
    mint ret = 0;
    for (int i = 0; i <= a.deg(); i++) ret += a[i] * d[i];
    return ret;
}
```

1.7 Mod Int

Should be **added**.

1.8 Miller Rabin w/ Pollard Rho

Should be **revised**.

```
ll mult(ll a, ll b, ll mod) {
    return a * b % mod;
}

ll power(ll base, ll exp, ll mod) {
    ll result = 1;
    while (exp > 0) {
        if (exp & 1) result = mult(result, base, mod);
        base = mult(base, base, mod);
        exp >>= 1;
    }
    return result;
}

bool miller_rabin(ll n, ll d, ll r, ll a) {
    ll x = power(a, d, n);
    if (x == 1 || x == n - 1) return true;
    for (int i = 1; i < r; i++) {
        x = mult(x, x, n);
        if (x == n - 1) return true;
    }
    return false;
}

bool is_prime(ll n) {
    if (n == 2 || n == 3) return true;
    if (n <= 1 || !(n & 1)) return false;

    ll d = n - 1, r = 0;
    while (!(d & 1)) {
        d >>= 1;
```

```
        r++;
    }

    int test[] = {2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37};
    for (const int &i: test) {
        if (i > n - 2) break;
        if (!miller_rabin(n, d, r, i)) return false;
    }
    return true;
}

ll pollard_rho(ll n) {
    if (!(n & 1)) return 2;
    ll x = rand() % (n - 2) + 2, y = x, c = rand() % 10 + 1, d = 1;

    auto f = [&](ll x) { return (mult(x, x, n) + c) % n; };

    while (d == 1) {
        x = f(x);
        y = f(f(y));
        d = __gcd(abs(x - y), n);
    }
    return d;
}
```

1.9 Simplex
Should be added.
1.10 De Brujin Sequence
Should be added.
2 Geometry
2.1 Convex Hull
Should be added.
2.2 Dynamic Convex Hull
Should be added.
2.3 3D Convex Hull
Should be added.
2.4 Smallest Enclosing Sphere/Circle
Should be added.
2.5 K-D Tree
Should be added.
2.6 Half Plane Intersection
Should be added.
2.7 Delauney w/ Vornoi Diagram
Should be added.
3 Data Structure
3.1 Segment Tree w/ Lazy
Should be added.
3.2 Splay Tree
Should be added.
3.3 Link Cut Tree
Should be added.
3.4 Lichao Tree
Should be added.
4 Strings
4.1 Manacher
Should be added.
4.2 Hirschberg
Should be added.
4.3 Round LCS
Should be added.
4.4 Z
Should be added.
4.5 Suffix and LCP
Should be added.
4.6 Palindrome Tree
Should be added.
5 Graph
5.1 Dinic
Should be added.
5.2 SCC
Should be added.
5.3 2-SAT
Should be added.
5.4 Directed MST
Should be added.
5.5 SCC
Should be added.
5.6 Global Min Cut
Should be added.
5.7 General Matching
Should be added.
5.8 Centroid Decomposition
Should be added.
5.9 Bipartite Matching
Should be added.
5.10 Dominator Tree
Should be added.
5.11 MCMF
Should be added.
5.12 Gomory Hu Tree
Should be added.
5.13 BCC
Should be added.
5.14 Block Cut Tree
Should be added.
5.15 Cut Vertex/Edge
Should be added.

6 Misc.

6.1 Basic Template

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

using namespace std;
using ll = long long;
using ld = long double;
using ull = unsigned long long;
using vint = vector<int>;
using matrix = vector<vint>;
using vll = vector<ll>;
using matrlx = vector<vll>;
using pii = pair<int, int>;
using pll = pair<ll, ll>;
using vpii = vector<pii>;
using vpll = vector<pll>;
using dbl = deque<bool>;
using dbltrix = deque<dbl>;
using sint = stack<int>;
using tii = tuple<int, int, int>;
using vull = vector<ull>;

#define fastio ios::sync_with_stdio(false), cin.tie(NULL), cout.tie(NULL)
#define endl '\n'
#define _CRT_SECURE_NO_WARNINGS
#define all(vec) vec.begin(), vec.end()
#define rall(vec) vec.rbegin(), vec.rend()

const int INF = 0x3f3f3f3f;
const ll VINF = 2e18;
const double PI = acos(-1);
const int MOD = 998244353;

template <typename t>
istream& operator>>(istream& in, vector<t>& vec) {
    for (auto& x : vec) in >> x;
    return in;
}

template <typename t, typename u>
istream& operator>>(istream& in, pair<t, u>& i) {
    in >> i.first >> i.second;
    return in;
}
```

6.2 Primes

We Love Primes!

List of Evil Primes

```
709, 1493, 3209, 6427, 12983, 26267, 53201, 107897, 218971
```

List of DFT Friendly Primes

```
65537, 4294967291, 1000000007, 7516192771, 998244353
```