ICPC Templates

我们需要更深入浅出一些

Mine_King, Tx_Lcy, 369pai
洛谷科技

October 19, 2024

Contents

Chapter 1

字符串

1.1 最小表示法

```cpp
int i = 0, j = 1, k = 0;
while (k < n && i < n && j < n)
    if (a[(i + k) % n] == a[(j + k) % n]) k++;
    else {
        if (a[(i + k) % n] > a[(j + k) % n]) i = i + k + 1;
        else j = j + k + 1;
        if (i == j) i++;
        k = 0;
    }
ans = min(i, j);
```

1.2 Border 理论

1.2.1 关键结论

定理 1：对于一个字符串 s，若用 t 表示其最长的 Border，则有 $\mathcal{B}(s) = \mathcal{B}(t) \cup \{t\}$。

定理 2：一个字符串的 Border 与 Period 一一对应。具体地，$\text{pre}(s,i) \in \mathcal{B}(s) \iff |s| - i \in \mathcal{P}(s)$。

弱周期引理：

$$\forall p, q \in \mathcal{P}(s), p + q \le |s| \implies \gcd(p,q) \in \mathcal{P}(s)$$

定理 3：若字符串 t 是字符串 s 的前缀，且 $a \in \mathcal{P}(s), b \in \mathcal{P}(t), b \mid a, |t| \ge a$，且 b 是 t 的整周期，则有 $b \in \mathcal{P}(s)$。

周期引理：

$$\forall p,q \in \mathcal{P}(s), p+q-\gcd(p,q) \le |s| \implies \gcd(p,q) \in \mathcal{P}(s)$$

定理 4： 对于文本串 s 和模式串 t，若 $|t| \ge \frac{|s|}{2}$，且 t 在 s 中至少成功匹配了 3 次，则每次匹配的位置形成一个等差数列，且公差为 t 的最小周期。

定理 5： 一个字符串 s 的所有长度不小于 $\frac{|s|}{2}$ 的 Border 的长度构成一个等差数列。

定理 6： 一个字符串的所有 Border 的长度排序后可以划分成 $\lceil \log_2 |s| \rceil$ 个连续段，使得每段都是一个等差数列。

定理 7： 回文串的回文前/后缀即为该串的 Border。

定理 8： 若回文串 s 有周期 p，则可以把 $\text{pre}(s,p)$ 划分成长度为 $|s| \bmod p$ 的前缀和长度为 $p-|s| \bmod p$ 的后缀，使得它们都是回文串。

定理 9： 若 t 是回文串 s 的最长 Border 且 $|t| \ge \frac{|s|}{2}$，则 t 在 s 中只能匹配 2 次。

定理 10： 对于任意一个字符串以及 $u,v \in \text{Ssuf}(s), |u| < |v|$，一定有 u 是 v 的 Border。

定理 11： 对于任意一个字符串 s 以及 $u,v \in \text{Ssuf}(s), |u| < |v|$，一定有 $2|u| \le |v|$。

定理 12： $|\text{Ssuf}(s)| \le \log_2 |s|$。

1.2.2 KMP

```cpp
// n is |s|, m is |t|.
for (int i = 1, j = 0; i <= n; i++) {
    while (j && t[j + 1] != s[i]) j = pi[j];
    if (t[j + 1] == s[i])
        if (++j == m) {
            // ...
            j = nxt[j];
        }
}
```

1.3 Z 函数

```
// n is |s|.
for (int i = 2, j = 0; i <= n; i++) {
    if (i < j + z[j]) z[i] = min(z[i - j + 1], j + z[j] - i);
    while (i + z[i] <= n && s[i + z[i]] == s[1 + z[i]]) z[i]++;
    if (i + z[i] > j + z[j]) j = i;
}
```

1.4 Manacher

```
// t is the original string, n is |t|.
string s = "^#";
for (char i : t) s.push_back(i), s.push_back('#');
s.push_back ('@');
for (int i = 1, j = 0; i <= 2 * n + 1; i++) {
    if (i <= j + p[j]) p[i] = min (p[2 * j - i], j + p[j] - i);
    while (s[i - p[i] - 1] == s[i + p[i] + 1]) p[i]++;
    if (i + p[i] > j + p[j]) j = i;
}
```

1.5 AC 自动机

```
// == Preparations ==
#include <queue>
// == Main ==
struct ACAM {
    int tot, fail[200005], delta[200005][26];

    void insert(string s, int id) {
        int now = 0;
        for (char c : s) {
            int v = c - 'a';
            if (!delta[now][v]) delta[now][v] = ++tot;
            now = delta[now][v];
        }
        return;
    }
    void build() {
        queue<int> q;
        for (int c = 0; c < 26; c++)
            if (delta[0][c]) q.push(delta[0][c]);
        while (!q.empty()) {
            int now = q.front();
            q.pop();
            for (int c = 0; c < 26; c++)
                if (delta[now][c]) fail[delta[now][c]] = delta[fail[now]][c],
                 ↪  q.push(delta[now][c]);
                else delta[now][c] = delta[fail[now]][c];
        }
        return;
    }
} ac;
```

1.6 回文自动机 PAM

```
// == Main ==
struct PAM {
    int tot, delta[500005][26], len[500005], fail[500005], ans[500005];
    string s;
    int lst;

    PAM() {tot = 1; len[0] = 0; len[1] = -1; fail[0] = fail[1] = 1;}
    int getfail(int now, int i) {
        while (s[i - len[now] - 1] != s[i]) now = fail[now];
        return now;
    }
    void insert(int i) {
        int now = getfail(lst, i);
        if (!delta[now][s[i] - 'a']) {
            len[++tot] = len[now] + 2;
            fail[tot] = delta[getfail(fail[now], i)][s[i] - 'a'];
            delta[now][s[i] - 'a'] = tot;
            ans[tot] = ans[fail[tot]] + 1;
        }
        lst = delta[now][s[i] - 'a'];
        return;
    }
} p;
```

1.7 后缀自动机

1.7.1 普通 SAM

```
// == Main ==
struct SAM {
    int tot, lst;
    int len[2000005], siz[2000005], link[2000005];
    int delta[2000005][26];

    SAM() {link[0] = -1;}
    void insert(char ch) {
        int c = ch - 'a', now = ++tot;
        len[now] = len[lst] + 1;
        siz[now] = 1;
        for (int p = lst; p != -1; p = link[p])
            if (!delta[p][c]) delta[p][c] = tot;
            else if (len[delta[p][c]] == len[p] + 1) {link[now] = delta[p][c];
            ↪  break;}
            else {
                int q = delta[p][c], v = ++tot;
                len[v] = len[p] + 1;
                memcpy(delta[v], delta[q], sizeof(delta[v]));
                link[v] = link[q], link[q] = v, link[now] = v;
                for (int i = p; delta[i][c] == q; i = link[i]) delta[i][c] = v;
                break;
            }
        lst = now;
        return ;
    }
} sam;
```

1.7.2 广义 SAM

注意自动机空间要开 Trie 的两倍。

```
struct GSAM {
    int tot;
    int delta[2000005][26], link[2000005], len[2000005];
    struct Trie {
        int tot, trie[1000005][26], st[1000005];

        void insert(string s) {
            int now = 0;
            for (char c : s) {
                int id = c - 'a';
                if (!trie[now][id]) trie[now][id] = ++tot;
                now = trie[now][id];
            }
            return;
        }
    } tr;

    GSAM() {link[0] = -1;}
    int insert(int c, int lst) {
        int now = ++tot;
        len[now] = len[lst] + 1;
        for (int p = lst; p != -1; p = link[p])
            if (!delta[p][c]) delta[p][c] = now;
            else if (len[delta[p][c]] == len[p] + 1) {link[now] = delta[p][c];
            ↪ break;}
            else {
                int q = delta[p][c], v = ++tot;
                len[v] = len[p] + 1;
                memcpy(delta[v], delta[q], sizeof(delta[v]));
                link[v] = link[q], link[q] = v, link[now] = v;
                for (int i = p; i != -1 && delta[i][c] == q; i = link[i]) delta[i][c]
                ↪ = v;
                break;
            }
        return now;
    }
    void build() {
        queue<int> q;
        tr.st[0] = 0;
        q.push(0);
        while (!q.empty()) {
            int now = q.front();
            q.pop();
            for (int i = 0; i < 26; i++)
                if (tr.trie[now][i])
                    tr.st[tr.trie[now][i]] = insert(i, tr.st[now]),
                    ↪ q.push(tr.trie[now][i]);
        }
        return;
    }
} gsam;
```

1.8 后缀排序

```
// == Preparations ==
int sa[2000005], rk[2000005], b[1000005], cp[2000005];
// == Main ==
for (int i = 1; i <= n; i++) b[rk[i] = s[i]]++;
for (int i = 1; i < 128; i++) b[i] += b[i - 1];
for (int i = n; i >= 1; i--) sa[b[rk[i]]--] = i;
memcpy(cp, rk, sizeof(cp));
for (int i = 1, j = 0; i <= n; i++)
    if (cp[sa[i]] == cp[sa[i - 1]]) rk[sa[i]] = j;
    else rk[sa[i]] = ++j;
```

```
for (int w = 1; w < n; w <<= 1) {
    memcpy(cp, sa, sizeof(cp));
    memset(b, 0, sizeof(b));
    for (int i = 1; i <= n; i++) b[rk[cp[i] + w]]++;
    for (int i = 1; i <= n; i++) b[i] += b[i - 1];
    for (int i = n; i >= 1; i--) sa[b[rk[cp[i] + w]]--] = cp[i];
    memcpy(cp, sa, sizeof(cp));
    memset(b, 0, sizeof(b));
    for (int i = 1; i <= n; i++) b[rk[cp[i]]]++;
    for (int i = 1; i <= n; i++) b[i] += b[i - 1];
    for (int i = n; i >= 1; i--) sa[b[rk[cp[i]]]--] = cp[i];
    memcpy(cp, rk, sizeof(cp));
    for (int i = 1, j = 0; i <= n; i++)
        if (cp[sa[i]] == cp[sa[i - 1]] && cp[sa[i] + w] == cp[sa[i - 1] + w])
        ↪  rk[sa[i]] = j;
        else rk[sa[i]] = ++j;
}
```

Chapter 2

数论

2.1 Miller Rabin 和 Pollard Rho

2.1.1 Miller Rabin

```cpp
// == Preparations ==
const int prime[] = {2, 3, 5, 7, 9, 11, 13, 17, 19, 23, 29, 31, 37};

long long power(long long a, long long b, long long mod) {
    long long ans = 1;
    while (b) {
        if (b & 1) ans = (__int128)ans * a % mod;
        a = (__int128)a * a % mod;
        b >>= 1;
    }
    return ans % mod;
}
// == Main ==
inline int Miller_Rabin(long long n) {
    if (n == 1) return 0;
    if (n == 2) return 1;
    if (n % 2 == 0) return 0;
    long long u = n - 1, t = 0;
    while (u % 2 == 0) u /= 2, t++;
    for (int i = 0; i < 12; i++) {
        if (prime[i] % n == 0) continue;
        long long x = power(prime[i] % n, u, n);
        if (x == 1) continue;
        int flag = 0;
        for (int j = 1; j <= t; j++) {
            if (x == n - 1) {flag = 1; break;}
            x = (__int128)x * x % n;
        }
        if (!flag) return 0;
    }
    return 1;
}
```

2.1.2 Pollard Rho

```
// == Preparations ==
#include <chrono>
#include <random>

mt19937_64 gen(chrono::system_clock::now().time_since_epoch().count());
/*
Miller Rabin
*/
// == Main ==
long long Pollard_Rho(long long n) {
    long long s = 0, t = 0, c = gen() % (n - 1) + 1;
    for (int goal = 1; ; goal <<= 1, s = t) {
        long long val = 1;
        for (int step = 1; step <= goal; step++) {
            t = ((__int128)t * t + c) % n;
            val = (__int128)val * abs(t - s) % n;
            if (!val) return n;
            if (step % 127 == 0) {
                long long d = __gcd(val, n);
                if (d > 1) return d;
            }
        }
        long long d = __gcd(val, n);
        if (d > 1) return d;
    }
}
void factor(long long n) {
    if (n < ans) return ;
    if (n == 1 || Miller_Rabin(n)) {
        // n = 1 或 n 是一个质因子。
        // ...
    }
    long long p;
    do p = Pollard_Rho(n);
    while (p == n);
    while (n % p == 0) n /= p;
    factor(n), factor(p);
    return ;
}
```

2.2 常用数论算法

2.2.1 exgcd

求出来的解满足 $|x| \leq b, |y| \leq a$。

```cpp
// == Main ==
int exgcd(int a, int b, int &x, int &y) {
    if (b == 0) {x = 1; y = 0; return a;}
    int m = exgcd(b, a % b, y, x);
    y -= a / b * x;
    return m;
}
```

2.2.2 CRT

没用。

```cpp
// == Preparations ==
#include <vector>
// == Main ==
int CRT(vector<pair<int, int>> &a) {
    int M = 1;
    for (auto i : a) M *= i.second;
    int res = 0;
    for (auto i : a) {
        int t = inv(M / i.second, i.second);
        res = (res + (long long)i.first * (M / i.second) % M * t % M) % M;
    }
    return res;
}
```

2.2.3 exCRT

需保证 lcm 在 `long long` 范围内。

```cpp
// == Preparations ==
long long exgcd(long long a, long long b, long long &x, long long &y);
// == Main ==
long long exCRT(vector<pair<long long, long long>> vec) {
    long long ans = vec[0].first, mod = vec[0].second;
    for (int i = 1; i < (int)vec.size(); i++) {
        long long a = mod, b = vec[i].second, c = vec[i].first - ans % b;
        long long x, y;
        long long g = exgcd(a, b, x, y);
        if (c % g != 0) return -1;
        b /= g;
        x = (__int128)x * (c / g) % b;
        ans += x * mod;
        mod *= b;
        ans = (ans % mod + mod) % mod;
    }
    return ans;
}
```

2.2.4 exLucas

```
// == Preparations ==
int power(int a, long long b, int mod);
int exgcd(int a, int b, int &x, int &y);
int CRT(vector<pair<int, int>> &a);
// == Main ==
int inv(int n, int p) {
    int x, y;
    exgcd(n, p, x, y);
    return (x % p + p) % p;
}
int fac(long long n, int p, int pk) {
    if (n == 0) return 1;
    int res = 1;
    for (int i = 1; i < pk; i++)
        if (i % p != 0) res = (long long)res * i % pk;
    res = power(res, n / pk, pk);
    for (int i = 1; i <= n % pk; i++)
        if (i % p != 0) res = (long long)res * i % pk;
    return (long long)res * fac(n / p, p, pk) % pk;
}
int C(long long n, long long m, int p, int pk) {
    long long x = n, y = m, z = n - m;
    int res = (long long)fac(x, p, pk) * inv(fac(y, p, pk), pk) % pk * inv(fac(z, p,
     ↪  pk), pk) % pk;
    long long e = 0;
    while (x) e += x / p, x /= p;
    while (y) e -= y / p, y /= p;
    while (z) e -= z / p, z /= p;
    return (long long)res * power(p, e, pk) % pk;
}
int exLucas(long long n, long long m, int p) {
    vector<pair<int, int>> a;
    for (int i = 2; i * i <= p; i++)
        if (p % i == 0) {
            int pk = 1;
            while (p % i == 0) pk *= i, p /= i;
            a.emplace_back(C(n, m, i, pk), pk);
        }
    if (p != 1) a.emplace_back(C(n, m, p, p), p);
    return CRT(a);
}
```

Chapter 3

多项式

3.1 牛顿迭代

用于解决下列问题：

已知函数 G 且 $G(F(x)) = 0$，求多项式 F（ $\bmod x^n$）。

结论：

$$F(x) = F_*(x) - \frac{G(F_*(x))}{G'(F_*(x))} \pmod{x^n}$$

其中 $F_*(x)$ 为做到 $x^{n/2}$ 时的答案。

3.2 FFT

```cpp
// == Preparations ==
struct complex {
    double a, b;

    complex() = default;
    complex(double _a, double _b): a(_a), b(_b) {}
    complex operator+(const complex &x) const {return complex(a + x.a, b + x.b);}
    complex operator-(const complex &x) const {return complex(a - x.a, b - x.b);}
    complex operator*(const complex &x) const {return complex(a * x.a - b * x.b, a *
 ↪   x.b + b * x.a);}
    complex operator/(const complex &x) const {
        double t = b * b + x.b * x.b;
        return complex((a * x.a + b * x.b) / t, (b * x.a - a * x.b) / t);
    }
    complex &operator+=(const complex &x) {return *this = *this + x;}
    complex &operator-=(const complex &x) {return *this = *this - x;}
    complex &operator*=(const complex &x) {return *this = *this * x;}
    complex &operator/=(const complex &x) {return *this = *this / x;}
};
// == Main ==
void FFT(vector<complex> &f, int flag) const {
    int n = f.size();
    vector<int> swp(n);
    for (int i = 0; i < n; i++) {
        swp[i] = swp[i >> 1] >> 1 | ((i & 1) * (n >> 1));
        if (i < swp[i]) std::swap(f[i], f[swp[i]]);
    }
    for (int mid = 1; mid < n; mid <<= 1) {
        complex w1(cos(pi / mid), flag * sin(pi / mid));
        for (int i = 0; i < n; i += mid << 1) {
            complex w(1, 0);
            for (int j = 0; j < mid; j++, w *= w1) {
                complex x = f[i + j], y = w * f[i + mid + j];
                f[i + j] = x + y, f[i + mid + j] = x - y;
            }
        }
    }
    return;
}
```

3.3 常用 NTT 模数及其原根

模数	原根	分解
167772161	3	$5 \times 2^{25} + 1$
469762049	3	$7 \times 2^{26} + 1$
998244353	3	$119 \times 2^{23} + 1$
1004535809	3	$479 \times 2^{21} + 1$
2013265921	31	$15 \times 2^{27} + 1$
2281701377	3	$17 \times 2^{27} + 1$

3.4 多项式模板

```
// == Preparations ==
#include <vector>
// == Main ==
namespace Poly {
    const int mod = 998244353, G = 3, invG = 332748118;

    inline int power(int a, int b) {
        int ans = 1;
        while (b) {
            if (b & 1) ans = (long long)ans * a % mod;
            a = (long long)a * a % mod;
            b >>= 1;
        }
        return ans % mod;
    }

    struct poly: vector<int> {
        poly(initializer_list<int> &&arg): vector<int>(arg) {}
        template<typename... argT>
        poly(argT &&...args): vector<int>(forward<argT>(args)...) {}

        poly operator+(const poly &b) const {
            const poly &a = *this;
            poly ans(max(a.size(), b.size()));
            for (int i = 0; i < (int)ans.size(); i++)
                ans[i] = ((i < (int)a.size() ? a[i] : 0) + (i < (int)b.size() ? b[i]
                 ↪  : 0)) % mod;
            return ans;
        }
        poly operator+=(const poly &b) {return *this = *this + b;}
        poly operator-(const poly &b) const {
            const poly &a = *this;
            poly ans(max(a.size(), b.size()));
            for (int i = 0; i < (int)ans.size(); i++)
                ans[i] = ((i < (int)a.size() ? a[i] : 0) - (i < (int)b.size() ? b[i]
                 ↪  : 0) + mod) % mod;
            return ans;
        }
        poly operator-=(const poly &b) {return *this = *this - b;}
        void NTT(poly &g, int flag) const {
            int n = g.size();
            vector<unsigned long long> f(g.begin(), g.end());
            vector<int> swp(n);
            for (int i = 0; i < n; i++) {
                swp[i] = swp[i >> 1] >> 1 | ((i & 1) * (n >> 1));
                if (i < swp[i]) std::swap(f[i], f[swp[i]]);
            }
            for (int mid = 1; mid < n; mid <<= 1) {
                int w1 = power(flag ? G : invG, (mod - 1) / mid / 2);
                vector<int> w(mid);
                w[0] = 1;
                for (int i = 1; i < mid; i++) w[i] = (long long)w[i - 1] * w1 % mod;
                for (int i = 0; i < n; i += mid << 1)
                    for (int j = 0; j < mid; j++) {
                        int t = (long long)w[j] * f[i + mid + j] % mod;
                        f[i + mid + j] = f[i + j] - t + mod;
                        f[i + j] += t;
                    }
            if (mid == 1 << 10)
                for (int i = 0; i < n; i++) f[i] %= mod;
        }
        int inv = flag ? 1 : power(n, mod - 2);
        for (int i = 0; i < n; i++) g[i] = f[i] % mod * inv % mod;
        return;
    }

    // 下面是基于转置原理的 NTT，相对朴素版本效率更高。
    void NTT(poly &g, int flag) const {
        int n = g.size();
        vector<int> f(g.begin(), g.end());
        if (flag) {
            for (int mid = n >> 1; mid >= 1; mid >>= 1) {
                int w1 = power(G, (mod - 1) / mid / 2);
                vector<int> w(mid);
                w[0] = 1;
                for (int i = 1; i < mid; i++) w[i] = (long long)w[i - 1] * w1 % mod;
                for (int i = 0; i < n; i += mid << 1)
                    for (int j = 0; j < mid; j++) {
                        int t = (long long)(f[i + j] - f[i + mid + j] + mod) * w[j] % mod;
                        f[i + j] = f[i + j] + f[i + mid + j] >= mod ?
                            f[i + j] + f[i + mid + j] - mod : f[i + j] + f[i + mid + j];
                        f[i + mid + j] = t;
                    }
            }
            for (int i = 0; i < n; i++) g[i] = f[i];
        } else {
            for (int mid = 1; mid < n; mid <<= 1) {
                int w1 = power(invG, (mod - 1) / mid / 2);
                vector<int> w(mid);
                w[0] = 1;
                for (int i = 1; i < mid; i++) w[i] = (long long)w[i - 1] * w1 % mod;
                for (int i = 0; i < n; i += mid << 1)
                    for (int j = 0; j < mid; j++) {
                        int t = (long long)w[j] * f[i + mid + j] % mod;
                        f[i + mid + j] = f[i + j] - t < 0 ? f[i + j] - t + mod : f[i + j] - t;
                        f[i + j] = f[i + j] + t >= mod ? f[i + j] + t - mod : f[i + j] + t;
                    }
            }
            int inv = power(n, mod - 2);
            for (int i = 0; i < n; i++) g[i] = (long long)f[i] * inv % mod;
        }
```

```
            return;
        }

        poly operator*(poly b) const {
            poly a(*this);
            int n = 1, len = (int)(a.size() + b.size()) - 1;
            while (n < len) n <<= 1;
            a.resize(n), b.resize(n);
            NTT(a, 1), NTT(b, 1);
            poly c(n);
            for (int i = 0; i < n; i++) c[i] = (long long)a[i] * b[i] % mod;
            NTT(c, 0);
            c.resize(len);
            return c;
        }
        poly operator*=(const poly &b) {return *this = *this * b;}
        poly inv() const {
            poly f = *this, g;
            g.push_back(power(f[0], mod - 2));
            int n = 1;
            while (n < (int)f.size()) n <<= 1;
            f.resize(n << 1);
            for (int len = 2; len <= n; len <<= 1) {
                poly tmp(len), ff(len << 1);
                for (int i = 0; i < len >> 1; i++) tmp[i] = g[i] * 2 % mod;
                for (int i = 0; i < len; i++) ff[i] = f[i];
                g.resize(len << 1);
                NTT(g, 1), NTT(ff, 1);
                for (int i = 0; i < len << 1; i++) g[i] = (long long)g[i] * g[i] %
                 ↪  mod * ff[i] % mod;
                NTT(g, 0);
                g.resize(len);
                for (int i = 0; i < len; i++) g[i] = (tmp[i] - g[i] + mod) % mod;
            }
            g.resize(size());
            return g;
        }
        poly sqrt() const { // need F[0] = 1.
            poly f = *this, g;
            g.push_back(1);
            int n = 1;
            while (n < (int)f.size()) n <<= 1;
            f.resize(n << 1);
            for (int len = 2; len <= n; len <<= 1) {
                poly tmp(len), ff(len << 1);
                for (int i = 0; i < len >> 1; i++) tmp[i] = g[i] * 2 % mod;
                for (int i = 0; i < len; i++) ff[i] = f[i];
                g.resize(len << 1);
                NTT(g, 1);
                for (int i = 0; i < len << 1; i++) g[i] = (long long)g[i] * g[i] %
                 ↪  mod;
                NTT(g, 0);
                g += ff;
                g *= tmp.inv();
                g.resize(len);
            }
            g.resize(size());
            return g;
        }
        poly derivative() const {
            poly f(*this);
            for (int i = 1; i < (int)f.size(); i++) f[i - 1] = (long long)f[i] * i %
             ↪  mod;
            f.pop_back();
            return f;
        }
        poly integral() const {
            poly f(*this);
            f.push_back(0);
            for (int i = f.size() - 1; i >= 1; i--) f[i] = (long long)f[i - 1] *
             ↪  power(i, mod - 2) % mod;
            f[0] = 0;
            return f;
        }
        poly ln() const {
            poly f((derivative() * inv()).integral());
            f.resize(size());
            return f;
        }
        poly exp() const { // 需要满足 F[0] = 0
            poly f(*this), g;
            g.push_back(1);
            int n = 1;
            while (n < (int)size()) n <<= 1;
            f.resize(n);
            for (int len = 2; len <= n; len <<= 1) {
                poly tmp(g);
                g.resize(len);
                g = g.ln();
                for (int i = 0; i < len; i++) g[i] = (f[i] - g[i] + mod) % mod;
                g[0] = (g[0] + 1) % mod;
                g *= tmp;
                g.resize(len);
            }
            g.resize(size());
            return g;
        }
    };
```

```

    inline poly power(poly f, int b) { // 需要满足 F[0] = 1
        f = f.ln();
        for (int i = 0; i < (int)f.size(); i++) f[i] = (long long)f[i] * b % mod;
        f = f.exp();
        return f;
    }
    // 不要求 F[0] = 1 的多项式快速幂，但是我忘记怎么用了，记得去回顾一下！
    poly power(poly f, int b1, int b2 = -1) {
        if (b2 == -1) b2 = b1;
        int n = f.size(), p = 0;
        reverse(f.begin(), f.end());
        while (!f.empty() && !f.back()) f.pop_back(), p++;
        if (f.empty() || (long long)p * b1 >= n) return poly(n);
        int v = f.back();
        int inv = power(v, mod - 2);
        for (int &i : f) i = (long long)i * inv % mod;
        reverse(f.begin(), f.end());
        f = f.ln();
        for (int &i : f) i = (long long)i * b1 % mod;
        f = f.exp();
        reverse(f.begin(), f.end());
        for (int i = 1; i <= p * b1; i++) f.push_back(0);
        reverse(f.begin(), f.end());
        f.resize(n);
        v = power(v, b2);
        for (int &i : f) i = (long long)i * v % mod;
        return f;
    }
}
```

Chapter 4

杂项

4.1 取模类

```cpp
// == Main ==
struct mint {
    static const int mod = 998244353;
    int v;

    mint() = default;
    mint(int _v): v((_v % mod + mod) % mod) {}
    explicit operator int() const {return v;}
    mint operator+(const mint &x) const {return v + x.v - (v + x.v < mod ? 0 : mod);}
    mint &operator+=(const mint &x) {return *this = *this + x;}
    mint operator-(const mint &x) const {return v - x.v + (v - x.v >= 0 ? 0 : mod);}
    mint &operator-=(const mint &x) {return *this = *this - x;}
    mint operator*(const mint &x) const {return (long long)v * x.v % mod;}
    mint &operator*=(const mint &x) {return *this = *this * x;}
    mint inv() const {
        mint a(*this), ans(1);
        int b(mod - 2);
        while (b) {
            if (b & 1) ans *= a;
            a *= a;
            b >>= 1;
        }
        return ans;
    }
    mint operator/(const mint &x) const {return *this * x.inv();}
    mint &operator/=(const mint &x) {return *this = *this / x;}
    mint operator-() {return mint(-v);}
};
```

4.1.1 Barrett 约减

当模数不固定时可以加速。

用法：在构造函数中传模数，使用方法为 `F.reduce(x)`，其中 x 是需要取模的数。

```cpp
// == Main ==
struct Barrett {
    unsigned long long b, m;
    Barrett(unsigned long long b = 2): b(b), m((__uint128_t(1) << 64) / b) {}
    unsigned long long reduce(long long x) {
        unsigned long long r = (__uint128_t(x + b) * m) >> 64;
        unsigned long long q = (x + b) - b * r;
        return q >= b ? q - b : q;
    }
} F;
```

4.2 对拍脚本

```bash
#!/usr/bin/bash

declare -i num=0

while [ true ]; do
    ./mkdata > in.txt
    time ./mine < in.txt > out.txt
    ./correct < in.txt > ans.txt
    diff out.txt ans.txt
    if [ $? -ne 0 ]; then
        echo "WA"
        break
    fi
    num=num+1
    echo "Passed $num tests."
done
```

4.3 VS Code 配置

4.3.1 User Tasks

```json
{
    // See https://go.microsoft.com/fwlink/?LinkId=733558
    // for the documentation about the tasks.json format
    "version": "2.0.0",
    "tasks": [
        {
            "type": "shell",
            "label": "My C++ Runner",
            "detail": "Build and Run Current C++ Program",
            "command": [ // 三个编译方式保留一个即可。
                "clear",
                "&&",
                "g++ ${file} -o ${fileDirname}/${fileBasenameNoExtension} -std=c++14 -Wall -Wextra && echo '== Normal =='",
                "g++ ${file} -o ${fileDirname}/${fileBasenameNoExtension} -std=c++14 -Wall -Wextra -O2 && echo '== O2 =='",
                "g++ ${file} -o ${fileDirname}/${fileBasenameNoExtension} -std=c++14 -Wall -Wextra -fsanitize=undefined,address && echo '== UB Check =='",
                "&&",
                "gnome-terminal -- bash -c \"ulimit -s 524288; time ${fileDirname}/${fileBasenameNoExtension}; read -p 'Press ENTER to continue...'; exit\""
            ],
            "problemMatcher": [ // 非必要
                "$gcc"
            ],
            "group": { // 非必要
                "kind": "build",
                "isDefault": true
            },
            "presentation": { // 非必要
                "showReuseMessage": false
            }
        }
    ]
}
```

4.3.2 设置

- 字体大小：16。（`"editor.fontSize": 16`）
- 添加多个光标的方式：ctrl。（`"editor.multiCursorModifier": "ctrlCmd"`）
- 不适用空格代替 Tab。（`"editor.insertSpaces": false`）
- 不允许 Enter 进行代码补全。（`"editor.acceptSuggestionOnEnter": "off"`）
- 标尺：110。（`"editor.rulers": [110]`）
- 平滑。（`"editor.cursorSmoothCaretAnimation": "on"`）

- 标题栏外观。（`"window.titleBarStyle": "custom"`）

totally：

```json
{
    "editor.fontSize": 16,
    "editor.multiCursorModifier": "ctrlCmd",
    "editor.insertSpaces": false,
    "editor.acceptSuggestionOnEnter": "off",
    "editor.rulers": [110],
    "editor.cursorSmoothCaretAnimation": "on",
    "window.commandCenter": false
}
```

4.3.3 快捷键

- 切换块注释：Ctrl+Shift+A -> Ctrl+Shift+/
- 运行任务：Ctrl+Shift+B -> F11
- 向上移动行：Alt+up -> Ctrl+Shift+up
- 向下移动行：Alt+down -> Ctrl+Shift+down