cycleke（菜鸡）的 XCPC 模板

哈爾濱工業大學

cycleke

February 7, 2021

Contents

1 数学

1.1 素数

素数的数目有近似 $\pi(x) \sim \frac{x}{\ln(x)}$，判定如下:

```
inline ll mmul(const ll &a, const ll &b, const ll &mod) {
  ll k = (ll)((1.0L * a * b) / (1.0L * mod)), t = a * b - k * mod;
  for (t -= mod; t < 0; t += mod) {}
  return t;
}
inline ll mpow(ll a, ll b, const ll &mod) {
  ll res = 1;
  for (; b; b >>= 1, a = mmul(a, a, mod)) (b & 1) && (res = mmul(res, a, mod));
  return res;
}

inline bool check(const ll &x, const ll &p) {
  if (!(x % p) || mpow(p % x, x - 1, x) ^ 1) return false;
  for (ll k = x - 1, t; ~k & 1;) {
    if (((t = mpow(p % x, k >>= 1, x)) ^ 1) && (t ^ (x - 1))) return false;
    if (!(t ^ (x - 1))) return true;
  }
  return true;
}

inline bool Miller_Rabin(const ll &x) {
  if (x < 2) return false;
  static const int p[12] = {2, 3, 5, 7, 11, 13, 17, 19, 61, 2333, 4567, 24251};
  for (int i = 0; i < 12; ++i) {
    if (!(x ^ p[i])) return true;
    if (!check(x, p[i])) return false;
  }
  return true;
}
```

1.2 Pollard Rho

```
mt19937_64 rnd(chrono::high_resolution_clock::now().time_since_epoch().count());
inline ll rand64(ll x) { return rnd() % x + 1; }

inline ll Pollard_rho(const ll &x, const int &y) {
  ll v0 = rand64(x), v = v0, d, s = 1;
  for (int t = 0, k = 1;;) {
    v = (mmul(v, v, x) + y) % x, s = mmul(s, abs(v - v0), x);
    if (!(v ^ v0) || !s) return x;
    if (++t == k) {
      if ((d = __gcd(s, x)) ^ 1) return d;
      v0 = v, k <<= 1;
    }
  }
}

vector<ll> factor;
void findfac(ll n) {
  if (Miller_Rabin(n)) {
    factor.push_back(n);
    return;
  }
  ll p = n;
  while (p >= n) p = Pollard_rho(p, rand64(n));
  findfac(p), findfac(n / p);
}
```

1.3 欧拉函数

欧拉函数的性质:

- 欧拉函数是积性函数;
- $n = \sum_{d|n} \varphi(d)$;
- 若 $n = p^k$,其中 p 是质数,那么 $\varphi(n) = p^k - p^{k-1}$;
- 若 $\gcd(a, m) = 1$,则 $a^{\varphi(m)} \equiv 1 \pmod m$;

拓展欧拉定理:$a^b \equiv \begin{cases} a^{b \bmod \varphi(p)} & \gcd(a,,p) = 1 \\ a^b & \gcd(a,,p) \neq 1, b < \varphi(p) \\ a^{b \bmod \varphi(p) + \varphi(p)} & \gcd(a,,p) \neq 1,, b \geq \varphi(p) \end{cases} \quad (\pmod p)$

```
int euler_phi(int n) {
  int ans = n;
  for (int i = 2; i * i <= n; i++)
    if (n % i == 0) {
      ans = ans / i * (i - 1);
      while (n % i == 0) n /= i;
    }
  if (n > 1) ans = ans / n * (n - 1);
  return ans;
}
```

1.4 线性筛

factor 为最小质因子;mu 为莫比乌斯函数;phi 为欧拉函数;e 为质因子最高次幂,d 为因数个数;f 为因数和,g 为最小质因子的幂和,即 $p + p^1 + p^2 + \cdots + p^k$ 。理论上积性函数都可以线性筛。

```
const int MAXN = 1e7 + 5;
bitset<MAXN> vis;
int prime[MAXN / 15], prime_cnt;
int factor[MAXN], e[MAXN], d[MAXN], mu[MAXN], phi[MAXN];
void sieve() {
  factor[1] = 1, e[1] = 0, d[1] = 1, mu[1] = 1, phi[1] = 1;
  for (int i = 2; i < MAXN; ++i) {
    if (!vis[i]) {
      prime[prime_cnt++] = i;
      factor[i] = i;
      mu[i] = -1, phi[i] = i - 1;
      e[i] = 1, d[i] = 2;
      g[i] = f[i] = i + 1;
    }
    for (int j = 0, t; j < prime_cnt && (t = i * prime[j]) < MAXN; ++j) {
      vis[t] = 1;
      factor[t] = prime[j];
      if (i % prime[j] == 0) {
        mu[t] = 0, phi[t] = phi[i] * prime[j];
        e[t] = e[i] + 1, d[t] = d[i] / e[t] * (e[t] + 1);
        g[t] = g[i] * prime[j] + 1, f[t] = f[i] / g[i] * g[t];
        break;
      } else {
        mu[t] = -mu[i], phi[t] = phi[i] * (prime[j] - 1);
        e[t] = 1, d[t] = d[i] * 2;
        g[t] = 1 + prime[j], f[t] = f[i] * f[prime[j]];
      }
    }
  }
}
```

1.5 拓展欧几里得算法

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

1.6 莫比乌斯反演

常见积性函数（$f(ab)=f(a)f(b),(a,b)=1$）：

- 单位函数：$\epsilon(n)=[n=1]$ （完全积性）
- 恒等函数：$\mathrm{id}_k(n)=n^k$，$\mathrm{id}_1(n)$ 通常简记作 $\mathrm{id}(n)$（完全积性）。
- 常数函数：$1(n)=1$ （完全积性）
- 除数函数：$\sigma_k(n)=\sum_{d|n}d^k$，$\sigma_0(n)$ 通常简记作 $\mathrm{d}(n)$ 或 $\tau(n)$，$\sigma_1(n)$ 通常简记作 $\sigma(n)$。
- 欧拉函数：$\varphi(n)=\sum_{i=1}^{n}[\gcd(i,n)=1]$
- 莫比乌斯函数：$\mu(n)=\begin{cases}1 & n=1\\ 0 & \exists d>1,d^2\mid n\\ (-1)^{\omega(n)} & otherwise\end{cases}$，其中 $\omega(n)$ 表示 n 的本质不同质因子个数，它是一个加性函数（$\omega(ab)=\omega(a)+\omega(b)$）。

Dirichlet 卷积满足交换率、结合率和分配率，常见 Dirichlet 卷积：

- $\varepsilon=\mu*1$
- $d=1*1$
- $\sigma=\mathrm{id}*1$
- $\varphi=\mathrm{id}*\mu$
- $\mathrm{id}=\varphi*1$
- $\mathrm{id}^2=(\mathrm{id}\cdot\varphi)*\mathrm{id}$
- $f\cdot d=f*f$（f 为完全积性）

莫比乌斯反演：$f=g*1\iff g=\mu*f$。
拓展：对于数论函数 f,g 和完全积性函数 t 且 $t(1)=1$：

$$f(n)=\sum_{i=1}^{n}t(i)g\left(\left\lfloor\frac{n}{i}\right\rfloor\right)\iff g(n)=\sum_{i=1}^{n}\mu(i)t(i)f\left(\left\lfloor\frac{n}{i}\right\rfloor\right)$$

常用结论：

- $\sum_{i=x}^{n}\sum_{j=y}^{m}[\gcd(i,j)=k]=\sum_{d=1}\mu(d)\lfloor\frac{n}{kd}\rfloor\lfloor\frac{m}{kd}\rfloor$
- $d(ij)=\sum_{x|i}\sum_{y|j}[\gcd(x,y)=1]=\sum_{p|i,p|j}\mu(p)d\left(\frac{i}{p}\right)d\left(\frac{j}{p}\right)$

1.7 杜教筛

设 $S(n)=\sum_{i=1}^{n}f(i)$,则 $\sum_{i=1}^{n}(f*g)(i)=\sum_{i=1}^{n}g(i)S(\lfloor\frac{n}{i}\rfloor)\Rightarrow g(1)S(n)=\sum_{i=1}^{n}(f*g)(i)-\sum_{i=2}^{n}g(i)S(\lfloor\frac{n}{i}\rfloor)$。直接分块复杂度 $O(n^{\frac{3}{4}})$，预处理出前 $O(n^{\frac{2}{3}})$ 项复杂度为 $O(n^{\frac{2}{3}})$。常用结论：

- 莫比乌斯函数前缀和：$S(n)=\sum_{i=1}^{n}\epsilon(i)-\sum_{i=2}^{n}S(\lfloor\frac{n}{i}\rfloor)$
- 欧拉函数前缀和：$S(n)=\sum_{i=1}^{n}id(i)-\sum_{i=2}^{n}S(\lfloor\frac{n}{i}\rfloor)$

```
map<int, int> mp_mu;

int S_mu(int n) {
  if (n < MAXN) return sum_mu[n];
  if (mp_mu[n]) return mp_mu[n];
  int ret = 1;
  for (int i = 2, j; i <= n; i = j + 1) {
    j = n / (n / i);
    ret -= S_mu(n / i) * (j - i + 1);
  }
  return mp_mu[n] = ret;
}

// 使用莫比乌斯反演
ll S_phi(int n) {
  ll res = 0;
  for (int i = 1, j; i <= n; i = j + 1) {
    j = n / (n / i);
    res += 1LL * (S_mu(j) - S_mu(i - 1)) * (n / i) * (n / i);
  }
  return (res - 1) / 2 + 1;
}
```

1.8 类欧几里德算法

求 $f = \sum_{i=0}^{n} \lfloor \frac{ai+b}{c} \rfloor, g = \sum_{i=0}^{n} i \lfloor \frac{ai+b}{c} \rfloor, h = \sum_{i=0}^{n} \lfloor \frac{ai+b}{c} \rfloor^2$

```
const ll P = 998244353;
ll i2 = 499122177, i6 = 166374059;
struct data {
  data() { f = g = h = 0; }
  ll f, g, h;
}; // 三个函数打包
data calc(ll n, ll a, ll b, ll c) {
  ll ac = a / c, bc = b / c, m = (a * n + b) / c, n1 = n + 1, n21 = n * 2 + 1;
  data d;
  if (a == 0) { // 迭代到最底层
    d.f = bc * n1 % P;
    d.g = bc * n % P * n1 % P * i2 % P;
    d.h = bc * bc % P * n1 % P;
    return d;
  }
  if (a >= c || b >= c) { // 取模
    d.f = n * n1 % P * i2 % P * ac % P + bc * n1 % P;
    d.g = ac * n % P * n1 % P * n21 % P * i6 % P + bc * n % P * n1 % P * i2 % P;
    d.h = ac * ac % P * n % P * n1 % P * n21 % P * i6 % P +
          bc * bc % P * n1 % P + ac * bc % P * n % P * n1 % P;
    d.f %= P, d.g %= P, d.h %= P;

    data e = calc(n, a % c, b % c, c); // 迭代

    d.h += e.h + 2 * bc % P * e.f % P + 2 * ac % P * e.g % P;
    d.g += e.g, d.f += e.f;
    d.f %= P, d.g %= P, d.h %= P;
    return d;
  }
  data e = calc(m - 1, c, c - b - 1, a);
  d.f = n * m % P - e.f, d.f = (d.f % P + P) % P;
  d.g = m * n % P * n1 % P - e.h - e.f, d.g = (d.g * i2 % P + P) % P;
  d.h = n * m % P * (m + 1) % P - 2 * e.g - 2 * e.f - d.f;
  d.h = (d.h % P + P) % P;
  return d;
}
```

1.9 中国剩余定理

```
ll inv(ll a, ll p) {
  ll x, y;
  exgcd(a, p, x, y);
  return (x + p) % p;
}
ll CRT(ll n, ll *a, ll *m) {
  ll lcm = 1, res = 0;
  for (ll i = 0; i < n; ++i) lcm *= m[i];
  for (ll i = 0; i < n; ++i) {
    ll t = lcm / m[i], x = inv(t, m[i]);
    res = (res + a[i] * t % lcm * x) % lcm;
  }
  return res;
}
```

模数不互质的情况 $\begin{cases} x \equiv a_1 & (\pmod{m})_1) \\ x \equiv a_2 & (\pmod{m})_2) \end{cases}$，则转换为 $m_1 p - m_2 q = a_2 - a_1$，最终解（若有解）为 $x \equiv m_1 p + a_1 \pmod{\operatorname{lcm}(m_1, m_2)}$。

1.10 原根

- 阶：若 $(a, m) = 1$，使 $a^l \equiv 1 \pmod{m}$ 成立的最小的 l，称为 a 关于模 m 的阶，记为 $\mathrm{ord}_m a$。
- 原根：若 $(g, m) = 1, \mathrm{ord}_m g = \varphi(m)$，则称 g 为 m 的一个原根。若 m 有原根，则 m 一定是下列形式：$\{2, 4, p^a, 2p^a\}$（这里 p 为奇素数，a 为正整数）。
- 求原根：设 $p_1, p_2, \cdots, p_k$ 是 $\varphi(m)$ 的所有不同的素因数，则 g 是 m 的原根 $\iff \forall 1 \le i \le k$，有 $g^{\frac{\varphi(m)}{p_i}} \not\equiv 1 \pmod{m}$，集合 $S = g^s \mid 1 \le s \le \varphi(m), (s, \varphi(m)) = 1$ 给出 m 的全部原根。

1.11 BGSG

大步小步算法用来求离散对数 $x^k \equiv a \pmod{p}$。求解 $a^x \equiv b \pmod{p}$ 则，令 $x = A\lceil\sqrt{p}\rceil - B, 0 \le A, B \le \lceil\sqrt{p}\rceil$，有 $a^{A\lceil\sqrt{p}\rceil} \equiv ba^B \pmod{p}$，先枚举 A 之后在哈希表中查找 B 就行。

```
// Finds the primitive root modulo p
int generator(int p) {
  vector<int> fact;
  int phi = p - 1, n = phi;
  for (int i = 2; i * i <= n; ++i) {
    if (n % i == 0) {
      fact.push_back(i);
      while (n % i == 0) n /= i;
    }
  }
  if (n > 1) fact.push_back(n);
  for (int res = 2; res <= p; ++res) {
    bool ok = true;
    for (int factor : fact)
      if (mpow(res, phi / factor, p) == 1) {
        ok = false;
        break;
      }
    if (ok) return res;
  }
  return -1;
}
vector<int> BSGS(int n, int k, int a) {
  if (a == 0) return vector<int>({0});

  int g = generator(n);
  // Baby-step giant-step discrete logarithm algorithm
```

```cpp
  int sq = (int)sqrt(n + .0) + 1;
  vector<pair<int, int>> dec(sq);
  for (int i = 1; i <= sq; ++i)
    dec[i - 1] = {mpow(g, i * sq * k % (n - 1), n), i};

  sort(dec.begin(), dec.end());
  int any_ans = -1;
  for (int i = 0; i < sq; ++i) {
    int my = mpow(g, i * k % (n - 1), n) * a % n;
    auto it = lower_bound(dec.begin(), dec.end(), make_pair(my, 0));
    if (it != dec.end() && it->first == my) {
      any_ans = it->second * sq - i;
      break;
    }
  }
  if (any_ans == -1) return vector<int>();
  // Print all possible answers
  int delta = (n - 1) / __gcd(k, n - 1);
  vector<int> ans;
  for (int cur = any_ans % delta; cur < n - 1; cur += delta)
    ans.push_back(mpow(g, cur, n));
  sort(ans.begin(), ans.end());
  return ans;
}
```

1.12 自适应 Simpson

计算 $\int_a^b f(x)dx$。

```cpp
double simpson(double a, double b) {
  double c = a + (b - a) / 2;
  return (f(a) + 4 * f(c) + f(b)) * (b - a) / 6;
}
double integral(double a, double b, double eps, double A) {
  double c = a + (b - a) / 2;
  double L = simpson(a, c), R = simpson(c, b);
  if (fabs(L + R - A) <= 15 * eps) return L + R + (L + R - A) / 15;
  return integral(a, c, eps / 2, L) + integral(c, b, eps / 2, R);
}
double integral(double a, double b, double eps) {
  return integral(a, b, eps, simpson(a, b));
}
```

1.13 卢卡斯定理

卢卡斯定理：$\binom{n}{m} \bmod p = \binom{\lfloor n/p \rfloor}{\lfloor m/p \rfloor} \cdot \binom{n \bmod p}{m \bmod p} \bmod p$

```cpp
ll lucas(ll n, ll m, int p) {
  ll ret = 1;
  while (n && m) {
    ll nn = n % p, mm = m % p;
    if (nn < mm) return 0;
    ret = ret * fac[nn] % p * inv_fac[mm] % p * inv_fac[nn - mm] % p;
    n /= p, m /= p;
  }
  return ret;
}
```

拓展卢卡斯定理：用于处理 p 不是质数的情况。

对于 $C_n^m \bmod p$，我们将其转化为 r 个形如 $a_i \equiv C_n^m \pmod{{q_i}^{\alpha_i}}$ 的同余方程并分别求解；对于 $a_i \equiv C_n^m \pmod{{q_i}^{\alpha_i}}$，将 C_n^m 转化为 $\frac{\frac{n!}{q^x}}{\frac{m!}{q^y}\frac{(n-m)!}{q^z}} q^{x-y-z}$，可求逆元；对于 $\frac{m!}{q^y}$ 和 $\frac{(n-m)!}{q^z}$，将其变换整理，可递归求解。

```cpp
ll calc(ll n, ll x, ll p) {
  if (!n) return 1;
```

```
  ll s = 1;
  for (ll i = 1; i <= p; ++i) (i % x) && (s = s * i % p);
  s = mpow(s, n / p, p);
  for (ll i = n / p * p; i <= n; ++i) (i % x) && (s = s * (i % p) % p);
  return s * calc(n / x, x, p) % p;
}

ll multi_lucas(ll n, ll m, ll x, ll p) {
  ll cnt = 0;
  for (ll i = n; i; i /= x) cnt += i / x;
  for (ll i = m; i; i /= x) cnt -= i / x;
  for (ll i = n - m; i; i /= x) cnt -= i / x;
  return mpow(x, cnt, p) * calc(n, x, p) % p * inv(calc(m, x, p), p) % p *
         inv(calc(n - m, x, p), p) % p;
}

ll exlucas(ll n, ll m, ll P) {
  ll cnt = 0;
  static ll p[20], a[20];
  for (ll i = 2; i * i <= P; ++i) {
    if (P % i) continue;
    p[cnt] = 1;
    while (P % i == 0) p[cnt] *= i, P /= i;
    a[cnt] = multi_lucas(n, m, i, p[cnt]);
    ++cnt;
  }
  if (P > 1) p[cnt] = P, a[cnt] = multi_lucas(n, m, P, P), ++cnt;
  return CRT(cnt, a, p);
}
```