オイラー関数の多重合成を計算するアルゴリズム

Crimson Global Academy 梶田 光

概要

オイラー関数 $\varphi(n)$ の研究では, $\varphi(n)$ を含む方程式の解を研究する.
特に最近はオイラー関数の多重合成 $\varphi^k(n) := \underbrace{\varphi(...\varphi}_{k \text{ times}}(n)...)$ の研究が進み,
これを含む方程式の解を数値計算によって求めたい場面が多い.
しかし, $\varphi^k(n)$ は乗法的関数ではないので, 一般的に知られている区間篩をそのまま適用することができない.
そこで, 今回 3 つのアイデアから新しいアルゴリズムを作り, Rust で実装した.
以下, 正整数 k, N を固定し, $1 \le n \le N$ のすべての n に対して n の素因数分解と $\varphi(n), \varphi^2(n), ..., \varphi^k(n)$ が(メモリマップを用いて)高速に取得できるような LUT をディスク上に構築するアルゴリズムについて考える.

1. 部分分解とオイラー関数

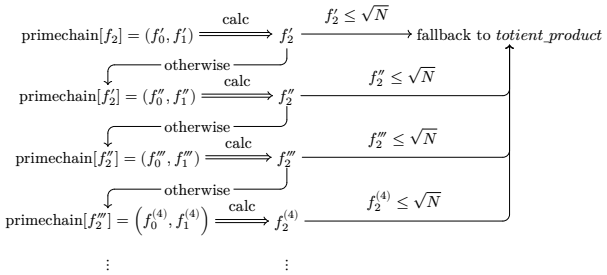
前回の発表では部分分解というアイデアを提案した.
これは正整数 $n \le N$ に対して $n = p_0 p_1 ... p_{m-1}$ $(p_i : \text{prime}, p_i \le p_{i+1}), i_0 = \max\left\{0 \le i \le m \mid \prod_{0 \le j < i} p_j \le \sqrt{N}\right\}, i_1 = \max\left\{i_0 \le i \le m \mid \prod_{i_0 \le j < i} p_j \le \sqrt{N}\right\}$ とおいたとき, $f_0 = \prod_{0 \le j < i_0} p_j, f_1 = \prod_{i_0 \le j < i_1} p_j, f_2 = \frac{n}{f_0 f_1}$ として $n = f_0 f_1 f_2$ という形に書き表す方法であり, これがメモリマップする区間を $O\left(\sqrt{N}\right)$ に抑えつつ素因数分解を取得するのに有用ということを示した.
しかし今回, これが $\varphi^k(n)$ を計算する際にも有用であることがわかった.
一般の n, m について, $d = \gcd(n, m)$ とすると $\varphi(nm) = \varphi(n)\varphi(m)\frac{d}{\varphi(d)}$ という公式がある.
しかしここで, $1 \le \varphi(n) \le n$ が成り立つことはもちろん, $1 \le \varphi(m)\frac{d}{\varphi(d)} \le m$ も成り立ち, さらにこれは自然数である.
したがって, $totient\_product([\alpha_0, \alpha_1, \alpha_2]) = \left[\varphi(\alpha_0), \varphi(\alpha_1)\frac{d}{\varphi(d)}, \varphi(\alpha_2)\frac{d'}{\varphi(d')}\right]$ where $d = \gcd(\alpha_0, \alpha_1), d' = \gcd(\alpha_0\alpha_1, \alpha_2)$ のような関数を定義すれば, $\alpha = \alpha_0\alpha_1\alpha_2$ について $totient\_product([\alpha_0, \alpha_1, \alpha_2]) = [\alpha'_0, \alpha'_1, \alpha'_2], \alpha' = \alpha'_0\alpha'_1\alpha'_2$ とおいたとき $\varphi(\alpha) = \alpha'$ かつ各 $0 \le i \le 2$ について $\alpha'_i \le \alpha_i$ が成り立つ.
よって $\sqrt{N}$ 以下のすべての n について $\varphi(n)$ を計算しておけば, 任意の $f_0, f_1, f_2 \le \sqrt{N}$ を満たす $n = f_0 f_1 f_2$ について $[f_0, f_1, f_2]$ に先の関数 f を繰り返し適用することによって $\varphi(n), \varphi^2(n), ...$ を計算することができる.

2. $k = 2$ の場合: 調和級数のオーダーと空間計算量の見積り

まず $k = 2$, つまり $\varphi(n), \varphi^2(n)$ までしか計算しなくてよい場合を考える.
n の部分分解を $n = f_0 f_1 f_2$ とおいたとき, $f_0, f_1, f_2 \le \sqrt{N}$ ならば計算できることは先程示した.
$f_0, f_1 \le \sqrt{N}$ は定義から直ちに従うので, $f_2 > \sqrt{N}$ の場合を考える.
前回示した定理より, f_2 は素数であるから, $\alpha = f_0 f_1 \le \sqrt{N}$ とおくと $\varphi(n) = \varphi(\alpha)(f_2 - 1)$.
ここから $\varphi^2(n)$ を計算するには $f_2 - 1$ の分解の情報が必要である.
そこで, primechain という長さ N の配列を用意する.
区間篩が $\sqrt{N} < [\text{start}, \text{end}]$ の区間で実行されているとすると, primechain の $\left[\frac{\text{start}}{2}, \frac{\text{end}}{2}\right], \left[\frac{\text{start}}{3}, \frac{\text{end}}{3}\right], \left[\frac{\text{start}}{4}, \frac{\text{end}}{4}\right], ..., \left[\frac{\text{start}}{\sqrt{N}}, \frac{\text{end}}{\sqrt{N}}\right]$ の部分をメモリマップする. (この方法を harmonic map と呼ぶことにする.)
$\text{start} \le n \le \text{end}$ かつ n が素数 (つまり $f_0 = f_1 = 1$)であれば, $n - 1 = f_2 - 1 = f'_0 f'_1 f'_2$ と部分分解したときの $[f'_0, f'_1]$ を primechain$[n]$ に記録する.
その後, $f_2 > \sqrt{N}$ で $totient\_product$ の繰り返しによって $\varphi^2(n)$ を計算できない場合, primechain$[f_2]$ から $f_2 - 1 = f'_0 f'_1 f'_2$ と部分分解したときの $[f'_0, f'_1]$ を取得.
f'_2 がそこから計算でき, これが $\sqrt{N}$ 以下なら $\varphi(n)$ は $\varphi(\alpha) \cdot f'_0 \cdot f'_1 \cdot f'_2$ と $\sqrt{N}$ 以下の正整数の積で表せる.
そうでなければ, f'_2 は素数なので $\varphi(n) = \varphi(\varphi(\alpha) \cdot f'_0 \cdot f'_1) \cdot (f'_2 - 1)$ と計算すればよい.
ここでメモリマップする範囲は調和級数の発散する速度より $O((\text{end} - \text{start}) \log N)$ 程度にしかならず, 区間の大きさを $\sqrt{N}$ 程度にとれば全体を通しての空間計算量は $O\left(\sqrt{N} \log N\right)$ ですむ.

3. 一般の場合: primechain の繋げ方

まず, n の分解 $n = f_0 f_1 f_2$ を考える.
そして, $f_2 > \sqrt{N}$ (ここから $f_2 : \text{prime}$ が従う) のときのみ $f_2 - 1$ の部分分解を $f_2 - 1 = f'_0 f'_1 f'_2$ と書く.
さらに, $f'_2 > \sqrt{N}$ (ここから $f'_2 : \text{prime}$ が従う) のときのみ $f'_2 - 1$ の部分分解を $f'_2 - 1 = f''_0 f''_1 f''_2$ と書く.
これを繰り返していき, $f''', f'''', ...$ を $f^{(3)}, f^{(4)}, ...$ と書くことにしよう.
さて, $\varphi(n)$ の計算には n の部分分解 $f_0 f_1 f_2$ が必要で, $\varphi^2(n)$ の計算には $f_2 - 1$ の部分分解 $f'_0 f'_1 f'_2$ が必要である.
厳密な議論は論文に書かれているが, $\varphi^3(n)$ の計算には $f'_2 - 1$ の部分分解 $f''_0 f''_1 f''_2$ が必要となり, 一般に $\varphi^k(n)$ の計算には $f_0^{(k-1)}, f_1^{(k-1)}, f_2^{(k-1)}$ までが必要である.
ここで, $f_2^{(i)}$ は n の約数ではないものの, 実はある簡単に計算できる整数 $B_i = \prod_{m=1}^{i} f_0^{(m)} f_1^{(m)}$ を用いて $f_2^{(i)} = \left\lfloor \frac{n}{B_i} \right\rfloor$ と書けることが証明できる. (GPT-5 との議論から発展, どのように GPT-5 が貢献したかは論文を参照のこと.)
つまり, f_2 だけでなく $f'_2, f''_2, ...$ はすべて harmonic map の範囲に含まれているから, primechain の harmonic map を $f'_0, f'_1, f'_2, f''_0, f''_1, f''_2, ...$ を取得する目的に使うことができる.



一般に $\varphi^k(n) = 1$ となる最小の k は $1 + \log_2 n$ で抑えられることが S. S. Pillai [1] によって示されているため, 実用上(オイラー関数の多重合成の性質を調べるという意味で)このアルゴリズムが使われるのは $k \le 1 + \log_2 N$ の範囲のみである.
これを考慮すると, 全体の時間計算量は $O(kN \log N)$, 空間計算量が $O\left(\sqrt{N} \log N\right)$, 消費するディスクの容量が $O(kN)$ となって, これは十分高速である.

最適化について

並列化とメモリマップの領域の最適化についても考察した.
特に後者については, primechain にアクセスするときの添字が素数であることから, wheel sieve の考え方を利用して空間計算量とディスクの容量を $\log \log N$ だけ落とすことができる.

テスト

他に RAM に入らない範囲の n の $\varphi^k(n)$ を計算するアルゴリズムが知られていないので, 既存のアルゴリズムとの速度比較はできない.
しかし, アルゴリズムの正当性のテストのため, RAM に入る範囲の $N = 10^8, k = 4$ で通常のエラトステネスの篩と比較し, $1 \le n \le N$ の範囲で $\varphi(n)$ から $\varphi^4(n)$ までの値がすべて一致することを確認した.

展望

今回のアルゴリズムはかなりオイラー関数固有の性質を活用しているため, 他に多重合成が研究されている乗法的関数 (約数の和関数など) にどこまで応用できるかはまだわかっていない.
また，本アルゴリズムを，時間計算量・空間計算量・ディスク使用量のいずれも悪化させずにさらに改善するのは困難に思われるが，この意味での最適性を示す厳密な証明または反例は現時点では得られていない.

リンク

論文: https://github.com/hikaru-kajita/mathematics/tree/main/multiphi-computation

参考文献

[1] S. S. Pillai, “On a function connected with ϕ (n),” Bulletin of the American Mathematical Society, vol. 35, no. 6, pp. 837–841, 1929.