ゼータ変換・メビウス変換

NOSS

2021/08/22

1 Introduction

2 畳み込み

a, b を長さ N の数列 $a = [a_0, a_1, \ldots, a_{N-1}], b = [b_0, b_1, \ldots, b_{N-1}]$ とする。ここで**畳み込み**とは、の 2 つの数列 a, b から数列 c を生成する操作であり、$c = a * b$ と表記する。畳み込みは、添字集合 $J = \{0, 1, \ldots, N-1\}$ 上の二項演算に応じていくつか考えられる。例として、

$$c_k = \sum_{i+j=k} a_i b_j \tag{1}$$

$$c_k = \sum_{ij=k} a_i b_j \tag{2}$$

$$c_k = \sum_{\max(i,j)=k} a_i b_j \tag{3}$$

$$c_k = \sum_{\min(i,j)=k} a_i b_j \tag{4}$$

$$c_k = \sum_{\gcd(i,j)=k} a_i b_j \tag{5}$$

$$c_k = \sum_{\mathrm{lcm}(i,j)=k} a_i b_j \tag{6}$$

$$c_k = \sum_{i \text{ or } j=k} a_i b_j \tag{7}$$

$$c_k = \sum_{i \text{ and } j=k} a_i b_j \tag{8}$$

$$c_k = \sum_{i \text{ xor } j=k} a_i b_j \tag{9}$$

$$\tag{10}$$

などが挙げられる。ただし、or, and, xor はそれぞれ bitwise-OR, bitwise-AND, bitwise-XOR である。

愚直な計算では、すべての (i, j) の組について演算するため、時間計算量は $O(N^2)$ となる。

3 ゼータ変換とメビウス変換

3.1 メビウスの反転公式

添字集合 J に半順序関係 $(\leq)$ が成り立ち、関数 $f(x)$ がすべての $x \in J$ に対して定義されているとする。このとき、f を**ゼータ変換**した関数 F は式 (11) で定義される。

$$F(n) = \sum_{x \leq n} f(x) \tag{11}$$

また、ゼータ変換の逆変換である**メビウス反転（メビウス変換）**は式 (12) で定義される。

$$f(n) = \sum_{x \leq n} \mu(x, n) F(x) \tag{12}$$

ここで、$\mu(n, x)$ はメビウス関数であり、式 (13) を満たす。

$$\sum_{m \leq x \leq n} \mu(x, n) = \delta(m, n) \tag{13}$$

$\delta(m, n)$ はクロネッカーのデルタであり、$\delta(n, n) = 1, \delta(m, n) = 0$ if $m \neq n$ である。

$$\begin{aligned}
\sum_{x \leq n} \mu(x, n) F(x) &= \sum_{x \leq n} \mu(x, n) \sum_{y \leq x} f(y) && (14)\\
&= \sum_{x \leq n} f(x) \sum_{x \leq y \leq n} \mu(y, n) && (15)\\
&= f(n) && (16)
\end{aligned}$$

3.1.1 具体例

添字集合に整数の大小関係による順序をおいて、$f = [3, 1, 4, 1, 5, 9, 2, 6]$ のゼータ変換を行うと、$F = [3, 4, 8, 9, 14, 23, 25, 31]$ となる。また、メビウス関数を、

$$\mu(m, n) = \begin{cases} 1 & m = n, \\ -1 & m = n - 1, \\ 0 & \text{otherwise.} \end{cases} \tag{17}$$

とおけば、メビウス変換が成り立つ。

$$[3, 1, 4, 1, 5, 9, 2, 6] \underset{\text{ゼータ変換}}{\overset{\text{メビウス変換}}{\leftrightarrows}} [3, 4, 8, 9, 14, 23, 25, 31]$$

これは式をよくみれば、f の累積和を求める操作そのものであるとわかる。そのため、直感的にはゼータ変換は累積和をとることに相当し、メビウス変換は差分をとることに相当すると思うと考えやすい。

3.1.2 アルゴリズム

ゼータ変換は累積和をとる操作なので、添え字が小さい方から順に足していけばよい。メビウス変換はゼータ変換の逆操作なので、手順を逆からたどればよい。計算量はともに $O(N)$ となる。

```
// zeta transform
for(int i=1; i<N; i++) f[i] += f[i-1];

// mobius transform
for(int i=N-1; i>0; i--) f[i] -= f[i-1];
```

3.2 ゼータ変換を利用した畳み込み

添え字集合 J が束であるとしたとき、結び $(\vee)$ を用いて以下のような畳み込み演算を定義できる。

$$(f * g)(n) = \sum_{x \vee y = n} f(x)g(y) \tag{18}$$

ここで、畳み込み演算は f, g のゼータ変換の点積に変形できる。

$$\begin{aligned} z\left((f * g)(n)\right) &= \sum_{x \vee y \leq n} f(x)g(y) \\ &= \sum_{\substack{x \leq n \\ y \leq n}} f(x)g(y) \\ &= \left(\sum_{x \leq n} f(x)\right)\left(\sum_{y \leq n} g(y)\right) \\ &= F(n)G(n) \end{aligned}$$

3.2.1 具体例

整数の大小関係による順序では、結び $\vee$ は max 演算に相当し、

$$(f *_{\max} g)(n) = \sum_{\max(x,y)=n} f(x)g(y) \tag{19}$$

となる。他にも自然数の整除関係を順序にとれば lcm の畳み込みになり、集合の包含関係を順序にとれば和集合 $\cup$ の畳み込みになる。

3.3 上位集合へのゼータ変換

ゼータ変換の別の定義として以下がある。

$$F(n) = \sum_{x \geq n} f(x) \tag{20}$$

すなわち、順序関係を逆転させた添え字集合に対してゼータ変換を定義している。ここではこれらを区別するために、式 (11) を**下位集合へのゼータ変換**、式 (20) を**上位集合へのゼータ変換**と呼ぶことにする。これらの定義の違いは、順序関係をどちらの方向にとるかによる違いである。

また、上位集合へのゼータ変換では交わり ($\wedge$) を用いた畳み込みを定義できる。

$$(f * g)(n) = \sum_{x \wedge y = n} f(x) g(y) \tag{21}$$

交わり ($\wedge$) 演算の例として、$\min, \gcd, \cap$ なとが挙げられる。

4 約数系ゼータ変換

4.1 メビウス関数

自然数 n に対して、メビウス関数 $\mu(n)$ を以下で定義する。

$$\mu(n) = \begin{cases} 1 & n = 1 \\ 0 & n \text{ がある素数の 2 乗で割り切れる} \\ (-1)^r & n \text{ が異なる } r \text{ 個の素数の積} \end{cases} \tag{22}$$

メビウス関数 $\mu(n)$ の値は，エラトステネスの篩と同様のアルゴリズムで前計算することができる．具体的には、素数 p の倍数 q を見ていったとき、

- q が p で 2 回以上割り切れる：$\mu(q) \leftarrow 0$
- そうでない場合：$\mu(q) \leftarrow -\mu(q)$

という更新をすべての素数について行えばよい。N 以下の自然数 n について $\mu(n)$ を前計算するとき、時間計算量は $O(N \log \log N)$ となる。

```
vector<int> mobius_table;

void build_mobius_func(int N){
    mobius_table.resize(N+1, 1);
    sieve(N);  // エラトステネスの篩で素数列挙

    for(int p=2; p<=N; p++){
        if(is_prime[p]){
            for(int q=p; q<=N; q+=p){
                if((q/p)%p == 0) mobius_table[q] = 0;
                else mobius_table[q] *= -1;
            }
        }
    }
}
```

4.2 ゼータ変換とメビウス変換

添え字集合を自然数の集合とし、順序に整除関係をとる。このときゼータ変換は

$$F(n) = \sum_{x|n} f(x) \tag{23}$$

メビウス変換は

$$f(n) = \sum_{x|n} \mu\left(\frac{n}{x}\right) F(x) \tag{24}$$

4.3 約数系の高速ゼータ変換

約数系ゼータ変換の愚直な計算量は，N 以下の k の倍数が N/k 個であることから、調和級数より $O(N \log N)$ となる。

約数系高速ゼータ変換は、多次元累積和をイメージすると理解しやすい。一般に自然数 n が N 以下の素数 $p_1, p_2, \ldots, p_m$ を用いて、

$$n = p_1^{e_1} p_2^{e_2} \cdots p_m^{e_m} \tag{25}$$

と素因数分解されるとき、これをベクトルで $(e_1, e_2, \ldots, e_m)$ と表記する。このとき、$x : (d_1, d_2, \ldots, d_m)$ が $x|n$ である条件は

$$d_1 \leq e_1, d_2 \leq e_2, \ldots, d_m \leq e_m \tag{26}$$

となる。したがって、ゼータ変換での $\sum_{x|n} f(x)$ の操作は多次元累積和をとることを意味する。すなわち、各素数 p_i ごとに e_i が小さい順から累積和をとることを行えばよい。

実際には、素数 p について $F(kp) \leftarrow F(kp) + f(k)$ の更新を k が小さい順に行う（p に対する軸方向の累積和)。$kp \leq N$ より、各素数についてループ回数は N/p のため、これはエラトステネスの篩の計算量と同じであることがわかる。したがって、時間計算量は $O(N \log \log N)$ となる。

```
template <class T>
vector<T> divisor_zeta_transform(vector<T> a){
    int N = a.size() - 1;
    for(int p=2; p<=N; p++){
        if(is_prime[p]){
            for(int k=1; k*p<=N; k++){
                a[k*p] += a[k];
            }
        }
    }
    return a;
}
```

4.4 約数系の高速メビウス変換

ゼータ変換の逆変換であるメビウス変換は、単純に逆から手順をたどればよい。アルゴリズム実装においてゼータ変換との差分は、ループを大きい方からまわす点と、符号 $(+/-)$ が反転している点である。

```
template <class T>
vector<T> divisor_mobius_transform(vector<T> a){
    int N = a.size()-1;
    for(int p=2; p<=N; p++){
        if(is_prime[p]){
            for(int k=N/p; k>=1; k--){
                a[k*p] -= a[k];
            }
        }
    }
    return a;
}
```

4.5 倍数系のゼータ変換

約数系の上位集合へのゼータ変換は以下のように定義される。

$$F(n) = \sum_{n|x} f(x) \tag{27}$$

5　部分集合のゼータ変換

ここでは、n 要素 $(n \geq 1)$ の集合 $N = \{0, 1, \ldots, n-1\}$ と、任意の部分集合 $S \subseteq N$ で定義される関数 $f(S)$ に対するゼータ変換について説明する。

5.1　ゼータ変換とメビウス変換

順序関係として集合の包含関係をとる場合、ゼータ変換は式 (28) で定義される。

$$F(S) = \sum_{X \subseteq S} f(X) \tag{28}$$

また、メビウス変換は式 (29) で定義される。

$$f(S) = \sum_{X \subseteq S} (-1)^{|S \setminus X|} F(X) \tag{29}$$

ここで、$A \setminus B$ は、A に属する元で B に属さないものからなる集合を表し、差集合と呼ばれる。

5.2　高速ゼータ変換

部分集合に対するゼータ変換の愚直な時間計算量は、部分集合列挙のテクニックを用いることで $O(3^n)$ となる。

部分集合の高速ゼータ変換についても、n 次元累積和として捉えると理解しやすい。すなわち、次元ごとに累積和をとっていけばよい。

ここで、$F_i(S)$ を、要素 i 未満までは S の部分集合、i 以上の要素は S と一致するものの総和、とする。意味的には、F_i は i 次元目までの累積和が完了した状態を表す。

$$F_i(S) = \sum_{X \in T} f(X),\ T = \{X | X \subseteq S, \forall j \geq i, X \cap \{j\} = S \cap \{j\}\} \tag{30}$$

アルゴリズムはまず初期化として、すべての部分集合 $S \subseteq N$ に対して

$$F_0(S) = f(S) \tag{31}$$

とする。続いて、$i = 0, 2, \ldots, n-1$ の順にすべての $S \subseteq N$ 対して以下の更新を行う。

$$F_{i+1}(S) = \begin{cases} F_i(S) & \text{if } i \notin S \\ F_i(S) + F_i(S \setminus \{i\}) & \text{if } i \in S \end{cases} \tag{32}$$

定義より、最後に $F(S) = F_n(S)$ としてゼータ変換が得られる。時間計算量は $O(n2^n)$ となる。実装は in-place な記述が可能である。

```
template <class T>
vector<T> subset_zeta_transform(vector<T> f, int n){
    for(int i=0; i<n; i++) {
        for(int S=0; S<(1<<n); S++) {
            if((S & (1<<i)) != 0) {  // if i in S
                f[S] += f[S^(1<<i)];
            }
        }
    }
    return f;
}
```

5.3 高速メビウス変換

ゼータ変換の逆変換であるメビウス変換は、単純に逆から手順をたどればよい。アルゴリズム実装においてゼータ変換との差分は、符号 $(+/-)$ が反転している点である。ループ順については、各 bit が 0,1 なので逆順にしなくてもよい。

アルゴリズムは以下のように記述できる。

$$f_0(S) = F(S) \tag{33}$$

$$f_{i+1}(S) = \begin{cases} f_i(S) & \text{if } i \notin S \\ f_i(S) - f_i(S \setminus \{i\}) & \text{if } i \in S \end{cases} \tag{34}$$

メビウス変換は $f(S) = f_n(S)$ で得られる。時間計算量は高速ゼータ変換と同様に $O(n2^n)$ となる。

```
template <class T>
vector<T> subset_mobius_transform(vector<T> f, int n){
    for(int i=0; i<n; i++) {
        for(int S=0; S<(1<<n); S++) {
            if((S & (1<<i)) != 0) {  // if i in S
                f[S] -= f[S^(1<<i)];
            }
        }
    }
    return f;
}
```

5.4 上位集合に対するゼータ変換

部分集合に対して上位集合に対するゼータ変換は、以下で定義される。

$$F(S) = \sum_{X \supseteq S} f(X) \tag{35}$$

また、メビウス変換は以下で定義される。

$$f(S) = \sum_{X \supseteq S} (-1)^{|X \setminus S|} F(X) \tag{36}$$

それぞれの高速ゼータ変換、高速メビウス変換は次のように実装できる。

```
template <class T>
vector<T> subset_zeta_transform(vector<T> f, int n){
    for(int i = 0; i < n; i++) {
        for(int S = 0; S < (1 << n); S++) {
            if((S & (1 << i)) == 0) {  // if i not in S
                f[S] += f[S ^ (1 << i)];
            }
        }
    }
    return f;
}

template <class T>
vector<T> subset_mobius_transform(vector<T> f, int n){
    for(int i=0; i<n; i++) {
        for(int S=0; S<(1<<n); S++) {
            if((S & (1<<i)) == 0) {  // if i not in S
                f[S] -= f[S^(1<<i)];
            }
        }
    }
    return f;
}
```

6 問題

6.1 AtCoder Beginner Contest 177 E - Coprime (500)

問題文

N 個の整数があります。i 番目の数は A_i です。「全ての $1 \leq i < j \leq N$ について、$\mathrm{GCD}(A_i, A_j) = 1$」が成り立つとき、$\{A_i\}$ は pairwise coprime であるといいます。 $\{A_i\}$ が pairwise coprime ではなく、かつ、 $\mathrm{GCD}(A_1, \ldots, A_N) = 1$ であるとき、 $\{A_i\}$ は setwise coprime であるといいます。 $\{A_i\}$ が pairwise coprime、setwise coprime、そのどちらでもない、のいずれであるか判定してください。 ただし GCD( ) は最大公約数を表します。

制約

- $2 \leq N \leq 10^6$
- $1 \leq A_i \leq 10^6$

サンプル 1

input : $N = 3, A = \{3, 4, 5\}$

output : pairwise coprime

サンプル 2

input : $N = 3, A = \{6, 10, 15\}$

output : setwise coprime

サンプル 3

input : $N = 3, A = \{6, 10, 16\}$

output : not coprime

6.2 AtCoder Beginner Contest 162 E - Sum of gcd of Tuples (Hard) (500)

問題文

1 以上 K 以下の整数からなる長さ N の数列 $\{A_1, ..., A_N\}$ を考えます。そのようなものは K^N 個ありますが、その全てについての $\gcd(A_1, ..., A_N)$ の和を求めてください。ただし、答えは非常に大きくなる可能性があるため、和を $(10^9 + 7)$ で割ったあまりを出力してください。なお、$\gcd(A_1, ..., A_N)$ は $A_1, ..., A_N$ の最大公約数を表します。

制約

- $2 \leq N \leq 10^5$
- $1 \leq K \leq 10^5$

サンプル 1

input : $N = 3, K = 2$

output : 9

6.3 CR 257 D. Jzzhu and Numbers

問題文

非負整数の集合 $a_1, ..., a_n$ の部分集合 S であって、それらの bitwise-AND が 0 であるような部分集合の個数を求めよ。

制約

- $1 \leq n \leq 10^6$
- $0 \leq a_i \leq 10^6$

サンプル 1

input : $N = 4, a = [0, 1, 2, 3]$

output : 10

7 解説

7.1 AtCoder Beginner Contest 177 E - Coprime (500)

setwise coprime であるかは全体の GCD が 1 であるかによって判定できるため、pairwise coprime の判定方法について考える。$f(n) = A$ に含まれる n の個数、$g(n) = \mathrm{GCD}(A_i, A_j)$ が n である (i, j) の個数、とおく。pairwise coprime である条件は、$A_i = 1$ のとき $\mathrm{GCD}(A_i, A_i) = 1$ であることに注意して、$g(1) = n(n-1) + f(1)$ と表せる。ここで、

$$g(n) = \sum_{\gcd(i,j)=n} f(i)f(j)$$

より、f, g の倍数系ゼータ変換をそれぞれ F, G とすれば、

$$G(n) = F(n) \cdot F(n)$$

となる。したがって、A_i の最大値を M として、$g(1)$ が高速ゼータ変換、高速メビウスで $O(M \log\log M)$ で求められる。全体の計算量は $O(N + M \log\log M)$ となる。

7.2 AtCoder Beginner Contest 162 E - Sum of gcd of Tuples (Hard) (500)

$f(n) = \gcd(A_1, ..., A_N)$ がちょうど n となる通り数、とおけば、求める解は、

$$\sum_{1 \leq x \leq K} f(x) \cdot x$$

となる。ここで、$F(n) = \gcd(A_1, ..., A_N)$ が n の倍数となる通り数、とすれば、

$$F(n) = \sum_{n|x} f(x) = \lfloor K/n \rfloor^N$$

より、F のメビウス変換から f が求められる。

8 参考文献

- https://noshi91.hatenablog.com/entry/2018/12/27/121649
- https://qiita.com/drken/items/3beb679e54266f20ab63