アダマール変換

Rogi

1 xor 畳み込み

1.1 問題

長さ 2^N の整数列 $a_0, a_1, \cdots, a_{2^N-1},\ b_0, b_1, \cdots, b_{2^N-1}$ が与えられるので、

$$c_k = \sum_{i \oplus j = k} a_i b_j$$

で定まる数列 $c_0, c_1, \cdots, c_{2^N-1}$ を求めよ。ただし、$i \oplus j$ は bitwise-XOR である。

この畳み込みを特に xor 畳み込み (Bitwise Xor Convolution) という。

Fact 1

サイズが 2 の FFT を考える。つまり、長さが 2 の整数列 $a_0, a_1,\ b_0, b_1$ が与えられて

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

を求める。多項式の積に帰着すると

$$(a_0 + a_1 x)(b_0 + b_1 x) = a_0 b_0 + (a_0 b_1 + a_1 b_0)x + a_1 b_1 x^2 = c_0 + c_1 x + c_2 x^2$$

求まる数列 c の長さは $2 + 2 - 1 = 3$ になるが、これを 2 に制限したらどうなるか。

つまり、$x^2 = 1$ として扱うと

$$
\begin{aligned}
(a_0 + a_1 x)(b_0 + b_1 x) &= (a_0 b_0 + a_1 b_1) + (a_0 b_1 + a_1 b_0)x \\
&= (a_0 b_0 + a_1 b_1)x^0 + (a_0 b_1 + a_1 b_0)x^1
\end{aligned}
$$

となって、xor の畳み込みができている。
この計算は、1 の 2 乗根 $(= -1, 1)$ を利用することでできる。

Fact 2

これまでに扱った DFT は 1 変数であった。多変数に拡張することを考える。

$$f(x, y) = 1 + 2x + 3y + 4xy$$

を考える。x, y の次数はどちらも 1 であるから、それぞれに 2 つの値を割り当てることで、$f(x, y)$ を決めることができる (多項式補間)。例えば、x に $10, 20$ を、y に $30, 40$ を割り当て、$f(10, 30), f(10, 40), f(20, 30), f(20, 40)$ を求めることができれば、$f(x, y)$ を復元できる。x と y に同じ値の組を割り当てることも可能で、$f(-1, 1), f(-1, -1), f(1, 1), f(1, -1)$ から $f(x, y)$ を復元できる。

また、多変数においても FFT を利用できる。f を

$$f(x, y) = (1 + 2x) + y(3 + 4x) = g_0(x) + g_1(y)$$

と y の次数でまとめる。すると、g_0, g_1 に FFT を適用して、$g_0(1), g_0(-1), g_1(1), g_1(0)$ を求める。すると

$$
\begin{aligned}
f(1, y) &= g_0(1) + g_1(1)y \\
f(-1, y) &= g_0(-1) + g_0(-1)y
\end{aligned}
$$

は 1 変数なので、FFT を適用できて、$f(-1, 1), f(-1, -1), f(1, 1), f(1, -1)$ が求まるので、FFT によって $f(x, y)$ を復元できた。
(cf: 多変数畳み込みの高速化
→ https://37zigen.com/truncated-multivariate-convolution/)

1.2 Xor Convolution

xor 畳み込みを多項式の積として表現するとき

$$ax^i \times bx^j \longrightarrow abx^{i \oplus j}$$

と計算したい。そこで、ビットごとに分け、多変数として考える。$i \oplus j$ の計算に必要な最大のビット数を考える必要があるが、ここでは説明のため 3 bit で考える。例えば

$$ax^3 \times bx^5 \xrightarrow{\text{bitwise}} ax_2^0 x_1^1 x_0^1 \times bx_2^1 x_1^0 x_0^1 \xrightarrow{\text{multiply}} abx_2^1 x_1^1 x_0^0 \xrightarrow{\text{bitwise}^{-1}} abx^6$$

$$3 \oplus 5 \xrightarrow{\text{bitwise}} 011 \oplus 101 \xrightarrow{\text{multiply}} 110 \xrightarrow{\text{bitwise}^{-1}} 6$$

とできる。これは多変数の畳み込みであり、かつ、それぞれのサイズは 2 であるから、Fact 1,2 から計算できることがわかった。

1.3 FFT ?

陽に FFT を利用せずとも計算が可能である。

$$f(x) = a + bx$$

について、$x = 1, -1$ を選ぶと

$$u = a + b$$
$$v = a - b$$

と変換できる。逆変換は

$$a = \frac{u+v}{2}$$
$$b = \frac{u-v}{2}$$

とできる。また、

$$u = \frac{a+b}{\sqrt{2}}$$
$$v = \frac{a-b}{\sqrt{2}}$$

と変換すれば

$$a = \frac{u+v}{\sqrt{2}}$$
$$b = \frac{u-v}{\sqrt{2}}$$

とできる。行列でこの変換を表現すると

$$\begin{bmatrix} u \\ v \end{bmatrix} = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} a \\ b \end{bmatrix}, \qquad \begin{bmatrix} a \\ b \end{bmatrix} = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix}$$

順変換と逆変換が一致する。

2　アダマール変換

2.1　クロネッカー積とクロネッカー冪

$n \times m$ 行列 A と任意サイズの行列 B のクロネッカー積を

$$A \otimes B := \begin{bmatrix} A_{11}B & A_{12}B & \cdots & A_{1m}B \\ \vdots & \vdots & \ddots & \vdots \\ A_{n1}B & A_{n2}B & \cdots & A_{nm}B \end{bmatrix}$$

と定義する。また、クロネッカー冪を

$$A^{\otimes n} := \underbrace{A \otimes \cdots \otimes A}_{n \text{ times}}$$

と定義する。

2.2　アダマール行列

アダマール行列 H_m を次のように定義する。

$$H_1 = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$$
$$H_m = H_1^{\otimes m}$$

また、次の定義もある。

$$H_0 = [1]$$
$$H_m = \frac{1}{\sqrt{2}} \begin{bmatrix} H_{m-1} & H_{m-1} \\ H_{m-1} & -H_{m-1} \end{bmatrix} = H_1 \otimes H_{m-1}$$

具体的には

$$H_1 = \frac{1}{\sqrt{2}^1} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$$

$$H_2 = \frac{1}{\sqrt{2}^2} \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \end{bmatrix}$$

$$H_3 = \frac{1}{\sqrt{2}^3} \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 & 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 & 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 & 1 & -1 & -1 & 1 \\ 1 & 1 & 1 & 1 & -1 & -1 & -1 & -1 \\ 1 & -1 & 1 & -1 & -1 & 1 & -1 & 1 \\ 1 & 1 & -1 & -1 & -1 & -1 & 1 & 1 \\ 1 & -1 & -1 & 1 & -1 & 1 & 1 & -1 \end{bmatrix}$$

また、アダマール行列 H_m の (i,j) 番目の要素は

$$(H_m)_{i,j} = \frac{1}{2^{m/2}}(-1)^{i \star j}$$

とかける。ただし、$i \star j$ はビットごとの内積を表すこととする。

2.3 xor 畳み込みとの関連

$f(x) = a + bx$ を変換するとき、

$$u = \frac{a+b}{\sqrt{2}}, \quad v = \frac{a-b}{\sqrt{2}}$$

と計算した。これを行列で記述すると

$$\begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} a \\ b \end{bmatrix} = H_1 \begin{bmatrix} a \\ b \end{bmatrix}$$

である。これを n 次元に拡張すると、H_n をかけることになる。

フーリエ変換する = アダマール行列をかける

2.4 畳み込み定理

関数 $f:\{0,1\}^n \to \mathbb{R}$ のフーリエ変換を

$$\widehat{f}(y) = \sum_{x\in\{0,1\}^n} f(x)(-1)^{x\star y}$$

と定義すると、これは、アダマール変換に他ならない。また、

$$\begin{aligned}
\widehat{g}(y)\widehat{h}(y) &= \left[\sum_{u\in\{0,1\}^n} g(u)(-1)^{u\star y}\right]\left[\sum_{v\in\{0,1\}^n} h(v)(-1)^{v\star y}\right] \\
&= \sum_{u\in\{0,1\}^n}\sum_{v\in\{0,1\}^n} g(u)h(v)(-1)^{u\star y+v\star y} \\
&= \sum_{u\in\{0,1\}^n}\sum_{v\in\{0,1\}^n} g(u)h(v)(-1)^{(u\oplus v)\star y} \\
&= \sum_{u\in\{0,1\}^n}\sum_{x\oplus u\in\{0,1\}^n} g(u)h(x\oplus u)(-1)^{x\star y} \\
&= \sum_{u\in\{0,1\}^n}\sum_{x\in\{0,1\}^n} g(u)h(x\oplus u)(-1)^{x\star y} \\
&= \sum_{x\in\{0,1\}^n}\left[\sum_{u\in\{0,1\}^n} g(u)h(x\oplus u)\right](-1)^{x\star y} \\
&= \sum_{x\in\{0,1\}^n} (g\circ h)(x)(-1)^{x\star y} \\
&= \widehat{g\circ h}(y)
\end{aligned}$$

$$\therefore \widehat{g}(y)\widehat{h}(y) = \widehat{g\circ h}(y)$$

を得る。最後に、フーリエ逆変換を

$$f(x) = \frac{1}{2^n}\sum_{y\in\{0,1\}^n} \widehat{f}(y)(-1)^{x\star y}$$

と定義できる。

$$
\begin{aligned}
\frac{1}{2^n}\sum_{y\in\{0,1\}^n}\widehat{f}(y)(-1)^{x\star y} &= \frac{1}{2^n}\sum_{y\in\{0,1\}^n}\left[\sum_{t\in\{0,1\}^n} f(t)(-1)^{t\star y}\right](-1)^{x\star y} \\
&= \frac{1}{2^n}\sum_{t\in\{0,1\}^n} f(t)\sum_{y\in\{0,1\}}(-1)^{(t\oplus x)\star y}
\end{aligned}
$$

ここで、$t = x$ のとき

$$
\sum_{y\in\{0,1\}}(-1)^{(t\oplus x)\star y} = \sum_{y\in\{0,1\}}(-1)^0 = 2^n
$$

$t \neq x$ のとき

$$
\sum_{y\in\{0,1\}}(-1)^{(t\oplus x)\star y} = 0
$$

であるから、

$$
\frac{1}{2^n}\sum_{t\in\{0,1\}^n} f(t)\sum_{y\in\{0,1\}}(-1)^{(t\oplus x)\star y} = \frac{1}{2^n}2^n f(x) = f(x)
$$

より、確かに逆変換になっていることがわかる。

2.5 Xor Convolution

数列 a, b の xor 畳み込みを求めたいとき

- a, b を多変数フーリエ変換 (mod 2) する
- 各要素の積を求める (c)
- c を多変数フーリエ逆変換 (mod 2) する

言い換えれば

- a, b をアダマール変換する
- 各要素の積 c を求める
- c をアダマール逆変換する

とすればよい。

3　高速アダマール変換

アダマール変換に限らず、クロネッカー冪行列 $G^{\otimes n}$ を 2^n 次元ベクトル x に掛ける計算 $G^{\otimes n}x$ を考える。ただし

$$G = \begin{bmatrix} a & b \\ c & d \end{bmatrix}$$

とする。このとき

$$\begin{aligned} G^{\otimes n} &= \begin{bmatrix} aG^{\otimes n-1} & bG^{\otimes n-1} \\ cG^{\otimes n-1} & dG^{\otimes n-1} \end{bmatrix} \\ &= \begin{bmatrix} aI_{n-1} & bI_{n-1} \\ cI_{n-1} & dI_{n-1} \end{bmatrix} \begin{bmatrix} G^{\otimes n-1} & 0 \\ 0 & G^{\otimes n-1} \end{bmatrix} \end{aligned}$$

この分割に基づいた分割統治法の計算量は $O(N \log N) = O(n2^n)$ となる。

```
template < class T >
void Kronecker(int n, vector<T> &x, T a, T b, T c, T d) {
    if(n == 0) return;
    int h = 1 << (n - 1);
    vector<T> l,r;
    for(int i = 0; i < h; i++) l.push_back(x[i]);
    for(int i = 0; i < h; i++) r.push_back(x[i + h]);
    Kronecker(n - 1, l, a, b, c, d);
    Kronecker(n - 1, r, a, b, c, d);
    for(int i = 0; i < h; i++) {
        T s = a * l[i] + b * r[i];
        T t = c * l[i] + d * r[i];
        x[i] = s;
        x[i + h] = t;
    }
}
```

2 つの行列は可換であるので、再帰呼出と for 文 を入れ替えてもよい。

また、バタフライ演算を用いて

```
template < class T >
void Kronecker(int n, vector<T> &x, T a, T b, T c, T d) {
    int m = 1 << n;
    for(int j = 1; j < m; j <<= 1) {
        for(int i = 0; i < m; i++) {
            if((i & j) == 0) {
                T s = a * x[i] + b * x[i + j];
                T t = c * x[i] + d * x[i + j];
                x[i] = s;
                x[i + j] = t;
            }
        }
    }
}
```

と書くこともできる。
バタフライ演算について → https://www.creativ.xyz/fast-fourier-transform/

4 ゼータ/メビウス変換について

係数行列を

$$\begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$$

とすると アダマール変換であった。

$$G = \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix}, \begin{bmatrix} 1 & 0 \\ -1 & 1 \end{bmatrix}$$

はそれぞれ ゼータ変換、メビウス変換に対応し、and と or の畳み込みを計算できる。

5 実装

xor 畳み込みを実装する。mod 998244353 で求める。

```
long long mod = 998244353;
void fwht(vector<long long> &a) {
    int n = (int)a.size();
    for(int d = 1; d < n; d <<= 1) {
        for(int m = d << 1, i = 0; i < n; i += m) {
            for(int j = 0; j < d; j++) {
                long long x = a[i + j], y = a[i + j + d];
                // xor
                    a[i + j] = (x + y) % mod;
                    a[i + j + d] = (x - y + mod) % mod;
                // and
                // a[i + j] = x + y;
                // or
                // a[i + j + d] = x + y;
            }
        }
    }
}

ll modinv(ll a){
    if(a==0) abort();
    ll b = mod, u = 1, v = 0;
    while(b){
        ll t = a/b;
        a -= t * b; swap(a,b);
        u -= t * v; swap(u,v);
    }
    u %= mod;
    if(u<0) u += mod;
    return u;
}


```

```
void ifwht(vector<long long> &a) {
    int n = (int)a.size();
    long long d2 = modinv(2);
    for(int d = 1; d < n; d <<= 1) {
        for(int m = d << 1, i = 0; i < n; i += m) {
            for(int j = 0; j < d; j++) {
                long long x = a[i + j], y = a[i + j + d];
                // xor
                    a[i + j] = (x + y) * d2 % mod;
                    a[i + j + d] = (x - y + mod) * d2 % mod;
                // and
                // a[i + j] = x - y;
                // or
                // a[i + j + d] = y - x;
            }
        }
    }
}

vector<long long> xor_convolution(vector<long long> &a, vector<long long> &b) {
    fwht(a); fwht(b);
    vector<long long> c(a.size());
    for(int i = 0; i < (int)a.size(); i++) c[i] = a[i] * b[i] % mod;
    ifwht(c); return c;
}

int main(){
    int N; cin >> N;
    vector<long long> a(1 << N), b(1 << N);
    for(int i = 0; i < (1 << N); i++) cin >> a[i];
    for(int i = 0; i < (1 << N); i++) cin >> b[i];
    vector<long long> c = xor_convolution(a, b);
    for(int i = 0; i < (1 << N); i++)
        cout << c[i] << " "; cout << endl;
}
```

verify : https://judge.yosupo.jp/problem/bitwise_xor_convolution

6 CS Academy Maxor (改題)

https://csacademy.com/contest/round-53/task/maxor

問題

長さ N の非負整数列 $A = \{A_1, A_2, \cdots, A_N\}$ が与えられる。
$A_i \oplus A_j$ (i, j は $1 \leq i < j \leq N$ を満たす整数) の最大値を求めよ。
また、最大値を達成する整数の組 (i, j) の数を求めよ。
(Bonus : 任意のビット演算)

制約

- $2 \leq N \leq 10^5$
- $0 \leq A_i < 2^{17}$

サンプル

input : $N = 4, \quad A = \{12, 3, 4, 11\}$
output : 最大値 15,　組の数 3

7 Codeforces Div.1 D. Little Pony and Elements of Harmony

https://codeforces.com/contest/453/problem/D

問題

$n = 2^m$ 個の宝石がある。宝石は完全グラフを形成し、互いに影響しあっている。
宝石は初め e_0 のエネルギーを持っている。宝石 u が エネルギーを $e_i[u]$ 持っているとき、その 1 秒後の宝石のエネルギー $e_{i+1}[u]$ は

$$e_{i+1}[u] = \sum_v e_{i-1}[v] \cdot b\left[\mathrm{bitcnt}(u \oplus v)\right]$$

で定まる。ただしここで、$b[\ ]$ は干渉係数であり、$m+1$ 個の要素からなる。また、$bitcnt(x)$ は x の立っているビットの数を表す。宝石の初めのエネルギー e_0 と干渉係数 b がわかっているので、t 秒後の宝石それぞれのエネルギーを p で割った余りを求めよ。

制約

- $1 \leq m \leq 20$
- $0 \leq t \leq 10^{18}$
- $2 \leq p \leq 10^9$
- $1 \leq e_0[i] \leq 10^9$
- $0 \leq b[i] \leq 10^9$

例

input :
m = 2,　t = 2,　p = 10000
$e_0 = 4, 1, 2, 3$
$b = 0, 1, 0$

output : $e_t = 14, 6, 6, 14$

8 Codeforces Div.1 E. Binary Table

https://codeforces.com/contest/663/problem/E

問題

$N \times M$ のグリッドが与えられる。それぞれのマスには 0 または 1 が書かれている。
一回の操作で次のいずれかを選択することができる。

- ある行の全てのマスについて 0 を 1 に、1 を 0 にする
- ある列の全てのマスについて 0 を 1 に、1 を 0 にする

有限回の操作で 1 があるマスの個数を最小化したい。達成できる最小値を求めよ。

制約

- $1 \leq n \leq 20$
- $1 \leq m \leq 10^5$

例

input :
$n = 3, \quad m = 4$
0110
1010
0111
output : 2

9 ABC 212 H - Nim Counting (600, difficulty = 2741)

https://atcoder.jp/contests/abc212/tasks/abc212_h

問題

正の整数 N, K と長さ K の整数列 $(A_1, A_2, \cdots, A_k)$ が与えられる。高橋君と青木君が Nim をする。ゲームを始める前の初期状態として次のようなものを考える。

- 山の個数を M は $1 \leq M \leq N$ を満たす
- 各山にある石の個数は $A_1, A_2, \cdots, A_K$ のいずれかである

初期状態として考えられるもののうち、2 人が自分が勝つために最適な行動をしたとき、高橋君が勝てるような初期状態の個数を 998244353 で割った余りを求めよ。ただし、山の順番を並び替えて一致するものは区別する。

制約

- $1 \leq N \leq 2 \times 10^5$
- $1 \leq K < 2^{16}$
- $1 \leq A_i < 2^{16}$
- $A_i \neq A_j$ for $1 \leq i < j \leq K$

例

input : $N = 2, \quad K = 2, \quad A = (1, 2)$
output : 4

10　参考

クロネッカー冪について

http://q.c.titech.ac.jp/docs/progs/kronecker.html

Wikipedia アダマール変換、高速アダマール変換

https://en.wikipedia.org/wiki/Fast_Walsh-Hadamard_transform

競技プログラミングにおける畳み込み問題まとめ アダマール変換

https://blog.hamayanhamayan.com/entry/2017/05/20/125607

Fast Walsh Hadamard Transforms and it's inner workings

https://codeforces.com/blog/entry/71899

練習問題

https://codeforces.com/contest/1464/problem/E

https://codeforces.com/problemset/problem/1218/D

https://codeforces.com/contest/1119/problem/H