
Свёртки

pakhomovee

Compiled: 31 июля 2023 г.

Часть I

Битовые Свёртки

1 Сумма по подмаскам

```
for (int bit = 0; bit < n; ++bit) {  
    for (int mask = 0; mask < (1 << n); ++mask) {  
        if ((1 << bit) & mask) {  
            dp[mask] += dp[mask ^ (1 << bit)];  
        }  
    }  
}
```

Доказательство.

Proposition 1.1

На шаге bit в $dp[mask]$ хранится сумма по подмаскам $mask$, где последние $(n - bit)$ битов такие же, как в $mask$.

Утверждение очевидно по индукции. 🧐

Несложно найти и SOS^{-1} :

```
for (int bit = 0; bit < n; ++bit) {  
    for (int mask = 0; mask < (1 << n); ++mask) {  
        if ((1 << bit) & mask) {  
            dp[mask] -= dp[mask ^ (1 << bit)];  
        }  
    }  
}
```

2 OR-свёртка

Definition 2.1

$SOS(a)$ - сумма по подмножествам массива a

Дано два массива a, b длины 2^n . Для каждого k от 0 до $2^n - 1$ посчитайте:

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

Proposition 2.2

Рассмотрим последовательность $h_i = \sum_{x \in i} \sum_{y \in i} a_x \times b_y = SOS(a)_i \times SOS(b)_i$.

Так как $x \in i, y \in i, x|y \in i$. Значит, $h = SOS(c)$, то есть $c = SOS^{-1}(h)$

3 AND-свёртка

Proposition 3.1. Fun Fact

Если $i \& j = k$, то $\bar{i}|\bar{j} = \bar{k}$

Отсюда получаем простой способ писать AND-свёртку: просто поменяем индексацию ($i \rightarrow (\bar{i})$), посчитаем *or*-свёртку, а потом опять поменяем индексацию.

4 Subset convolution

Definition 4.1. Замечание

Обозначим $SOS(array)$ за $zetaTransform(array)$, а $SOS^{-1}(array)$ за $muTransform(array)$. За $popcnt(x)$ обозначим количество единичных битов в двоичной записи числа x .

Дано два массива a, b длины 2^n . Для каждого k от 0 до $2^n - 1$ посчитайте:

$$\sum_{i|j=k, i \& j=0} a_i \times b_j$$

Proposition 4.2. Решение

Посчитаем $a'[i][j], b'[i][j]$ следующим образом ($0 \leq i \leq n, 0 \leq j < 2^n$):

- Если $i = popcnt(j)$, то $arr'[i][j] = arr[j]$
- Иначе $arr'[i][j] = 0$

Далее применим $zetaTransform$ ко всем строкам матрицы a' и b' (всего строк в матрице n). После этого вычислим матрицу res' :

$$res'[i][j] = \sum_{k=0}^i \sum_{j=0}^{2^n-1} a'[k][j] \times b'[i-k][j]$$

Применим $muTransform$ к строкам res' . Массив результата res определяется следующим образом:

$$res[i] = res'[popcnt(i)][i]$$

Доказательство.

Definition 4.3. Ranked Mobius Transform and Inversion

$\hat{f}(k, X) = \sum_{S \subseteq X, |S|=k} f(S)$. При этом инверсию $\hat{f}$ можно посчитать как $f(S) = \hat{f}(|S|, S)$. Однако более полезно записать это как $f(S) = \sum_{X \subseteq S} (-1)^{|S \setminus X|} \hat{f}(|S|, X)$.

Для функций $\hat{f}, \hat{g}$ определим

$$(\hat{f} \circ \hat{g})(k, X) = \sum_{j=0}^k \hat{f}(j, X) \times \hat{g}(k-j, X)$$

Proposition 4.4. Замечание

Обратите внимание, что свёртка происходит по рангу, а не по подмножествам.

Докажем, что, применяя к $(\hat{f} \circ \hat{g})$ инверсию, получаем subset convolution f и g .

$$\sum_{X \subseteq S} (-1)^{|S \setminus X|} (\hat{f} \circ \hat{g})(|S|, X) = \sum_{X \subseteq S} (-1)^{|S \setminus X|} \sum_{j=0}^{|S|} \hat{f}(j, X) \hat{g}(|S| - j, X) = \sum_{X \subseteq S} (-1)^{|S \setminus X|} \sum_{j=0}^{|S|} \sum_{\substack{U, V \subseteq X \\ |U|=j \\ |V|=|S|-j}} f(U)g(V)$$

Так как X пробегает все подмножества S , для любой упорядоченной пары (U, V) подмножеств S , что $|U| + |V| = |S|$, член $f(U)g(V)$ учитывается в сумме ровно один раз с коэффициентом $(-1)^{|S \setminus X|}$ для каждого X , что $U \cup V \subseteq X \subseteq S$. Другие слагаемые в сумме не учитываются. Получаем коэффициент при $f(U)g(V)$ равным

$$\sum_{x=|U \cup V|}^{|S|} \binom{|S| - |U \cup V|}{x - |U \cup V|} (-1)^{|S| - x} = \begin{cases} 1 & \text{если } |U \cup V| = |S| \\ 0 & \text{иначе} \end{cases}$$

Так как $|U| + |V| = |S|$ и $|U \cup V| = |S|$, выполнено $U \cup V = S, U \cap V = \emptyset$. Таким образом, мы научились искать subset convolution за $O(n^2 2^n)$. 🧐

5 XOR-свёртка

Мы хотим научиться искать по последовательностям a, b длины 2^n последовательность $c_i = \sum_{i=x \oplus y} a_x \times b_y$.

Proposition 5.1

Рассмотрим $hadamard(a)_i = \sum_j (-1)^{|i \& j|} \times a_j$, где $|i \& j|$ равно количеству битов 1 в пересечении масок i и j .

Мы хотим, как и до этого, показать, что:

$$hadamard(a) \times hadamard(b) = hadamard(c)$$

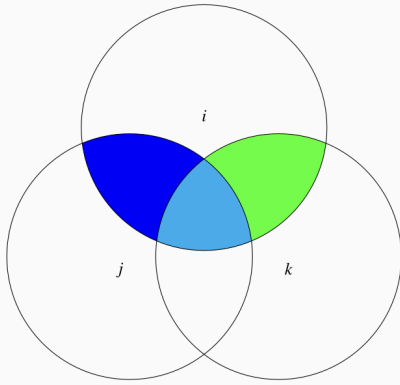
Для фиксированного i мы хотим доказать:

$$\left(\sum_j (-1)^{|i \& j|} \times a_j \right) \times \left(\sum_j (-1)^{|i \& j|} \times b_j \right) = \sum_j (-1)^{|i \& j|} \times c_j$$

$$\sum_{j, k} (-1)^{|i \& j| + |i \& k|} \times a_j \times b_k = \sum_j (-1)^{|i \& j|} \times c_j$$

Theorem 5.2

Доказательство. Пересечение i, j, k учитывается два раза и не влияет на чётность:



Таким образом, $hadamard(a)_i \times hadamard(b)_i = \sum_{j, k} (-1)^{|i \& (j \oplus k)|} \times a_j \times b_k$

Proposition 5.3

Обозначим $p := j \oplus k$

$$hadamard(a)_i \times hadamard(b)_i = \sum_p \sum_{j, k | j \oplus k = p} (-1)^{|i \& p|} \times a_j \times b_k$$

$$= \sum_p (-1)^{|i \& p|} \sum_{j, k | j \oplus k = p} a_j \times b_k = \sum_p (-1)^{|i \& p|} c_p = hadamard(c)_i$$

Теперь научимся считать $hadamard(a)$.

Proposition 5.4

$dp[mask][bit]$ — сумма по подмножествам $mask$, где меняются только первые bit битов, а знак при $a[submask]$ равен $(-1)^{|submask \& mask|}$.

Далее, при переходе от bit к $bit + 1$ нужно, возможно, поменять текущий бит, либо не менять его. Если текущий бит в маске равен нулю, то меняем мы его или нет, знаки остаются теми же самыми, потому что размер пересечения масок не изменился:

$$dp[mask][ind + 1] = dp[mask][ind] + dp[mask \oplus (1 \ll ind)][ind]$$

Если же текущий бит равен единице, и мы его не меняем, то в пересечение добавляется один новый бит, и знак меняется:

$$dp[mask][ind + 1] = dp[mask \oplus (1 \ll ind)][ind] - dp[mask][ind]$$

Это всё можно делать in-place:

```
vector<int> hadamard_transform(const vector<int>& a) {
    vector<int> dp = a;
    for (size_t bit = 1; bit < a.size(); bit <= 1) {
        for (size_t mask = 0; mask < a.size(); mask++) {
            if ((mask & bit) == 0) {
                int u = dp[mask], v = dp[mask ^ bit];
                dp[mask] = u + v;
                dp[mask ^ bit] = u - v;
            }
        }
    }
    return dp;
}
```

Обратное преобразование получить несложно:

Theorem 5.5

$$hadamard(hadamard(a))_i = 2^n \times a_i$$

Доказательство.

$$\begin{aligned} hadamard(hadamard(a))_i &= \sum_j (-1)^{|i \& j|} \times \left(\sum_k (-1)^{|j \& k|} \times a_k \right) = \sum_{j,k} (-1)^{|i \& j| + |j \& k|} a_k \\ &= \sum_{j,k} (-1)^{j \& (i \oplus k)} a_k = \sum_k a_k \times \left(\sum_j (-1)^{j \& (i \oplus k)} \right) \\ &= 2^n \times a_i \end{aligned}$$

Так как при $i \neq k$ вклад 0 (для каждой подмаски $i \oplus k$ значений j дающих вклад 1 столько же, сколько j дающих вклад -1 , при условии, что $k \& (i \oplus k)$ равно этой подмаске). 😎

6 gcd-свёртка

Мы хотим научиться искать по последовательностям a, b длины n последовательность $c_i = \sum_{\gcd(j,k)=i} a_j \times b_k$

Proposition 6.1

Рассмотрим $f(a)_i = \sum_{i|j} a_j$.

$$f(a)_i \times f(b)_i = \sum_{i|j, i|k} a_j \times b_k.$$

Заметим, что это в точности $f(c)_i$, так как $a_j \times b_k$ будет учтено в $c_{\gcd(a_j, b_k)}$.

Осталось только найти обратную свёртку к f . Это довольно понятная динамика, которую все уже 200 раз видели (как и эту задачу)!

```
void transform(vector<int> &a) {
    const int n = (int)a.size();
    for (int i = 1; i <= n; ++i) {
        for (int j = 2; j <= n / i; ++j) {
            a[i - 1] += a[i * j - 1];
        }
    }
}

void reverse_transform(vector<int> &a) {
    const int n = (int)a.size();
    for (int i = n; i >= 1; --i) {
        for (int j = 2; j <= n / i; ++j) {
            a[i - 1] -= a[i * j - 1];
        }
    }
}

vector<int> calc(vector<int> &a, vector<int> &b) {
    const int n = (int)a.size();
    vector<int> c(n);
    transform(a);
    transform(b);
    for (int i = 0; i < n; ++i) c[i] = a[i] * b[i];
    reverse_transform(c);
    return c;
}
```

7 lcm-свёртка

Мы хотим научиться искать по последовательностям a, b длины n последовательность $c_i = \sum_{\text{lcm}(j,k)=i} a_j \times b_k$

Proposition 7.1

Рассмотрим $f(a)_i = \sum_{j|i} a_j$.

$$f(a)_i \times f(b)_i = \sum_{j|i, k|i} a_j \times b_k.$$

Заметим, что это в точности $f(c)_i$, так как $a_j \times b_k$ будет учтено в $c_{\text{lcm}(j,k)}$

Осталось только найти обратную свёртку к f . Свёртка f — простая динамика, её легко обратить.