

# Побитови операции

Кирил Зулямски

## 1 Въведение

Побитовите операции са често срещани в състезателната информатика. Представяват операции върху две числа в двоична бройна система без преноси, тоест стойността на резултата на дадена позиция зависи единствено от битовете на тази позиция на операндите.

Срещат се три основни побитови операции: побитово и, побитово или и побитово изключващо или.

| $a$ | $b$ | $a \& b$ | $a   b$ | $a \oplus b$ |
|-----|-----|----------|---------|--------------|
| 0   | 0   | 0        | 0       | 0            |
| 0   | 1   | 0        | 1       | 1            |
| 1   | 0   | 0        | 1       | 1            |
| 1   | 1   | 1        | 1       | 0            |

За удобство често се използват и някои други побитови операции, но практически не се срещат като централна идея на задача:

- побитова инерсия ( $\sim x$ ) - обръща всички битове на дадено число;
- побитов ляв шифт ( $x \ll a$ ) - добавя  $a$  нули в края на двоичното представяне на  $x$ ; еквивалентно е на  $x \cdot 2^a$ ;
- побитов десен шифт ( $x \gg a$ ) - изтрива  $a$  цифри от края на двоичното представяне на  $x$ ; еквивалентно е на  $\frac{x}{2^a}$ ;

## 2 Употреба

- $\&$  и  $|$  не са обратими; при прилагане на  $\&$  (аналогично с  $|$ ) на дадено число или остава същото, или поне един бит изчезва (в частност не може да расте); това означава, че даден `int` може да се промени 32 пъти, преди да се занули.
- $\oplus$  е обратима и всъщност представлява събиране без пренос, тоест наследява всичките свойства на събирането (комунитативност, асоциативност, ...); това прави побитовото изключващо или значително мощна операция и доста по-често се среща в задачи

### 3 Задачи с големина

**Задача 1: (caterpillar3)** Да забележим, че от условието следва, че за самоописващ се подинтервал е вярно, че едно от числата е подмаска на всички други. Нека фиксираме най-лявото такова -  $a_i$ . Тогава трябва да преброим интервалите  $[l, r]$ , за които  $a_l, \dots, a_r$  са надмаски на  $a_i$ . Нека  $l_i$  е минимално, за което  $a_{l_i}, \dots, a_i$  са надмаски на  $a_i$ , и  $r_i$  е максимално, за което  $a_i, \dots, a_{r_i}$  са надмаски на  $a_i$ . Тогава лесно се вижда, че интервал  $[l, r]$  работи тогава и само тогава, когато  $l_i \leq l \leq i \leq r \leq r_i$ , тоест общият брой е  $(i - l_i + 1)(r_i - i + 1)$ . Остана да пресметнем бързо  $l_i$  и  $r_i$ , което става бързо с просто обхождане и поддържане на най-дясното срещане на всеки бит.

**Помощна задача:** Дадена е редица  $(a_i)_{i=1}^n$  и заявки, характеризиращи се с число  $x_j$ . За всяка заявка трябва да се върне най-голямата стойност от вида  $x_j \oplus a_i$ . Задачата се решава, като се фиксират битовете на отговора от най-старши към най-младши (след наблюдението, че най-старшият бит е най-значещ) - позиция на даден бит  $p$  на отговора е единица, ако при фиксиране на всичките битове на позиция поне  $p$  съществува подходящ елемент на множеството, за който може да се постигне този отговор.

**Задача 2: (888G)** За задачи с минимално покриващо дърво генерално има две опции - да използваме Крускал или Прим. В случая ще видим, че Прим трудно ще се навърже с постановката, затова ще използваме Крускал. Нека разгледаме последното добавено ребро, тоест максималното. Вярно ли е, че няма друго ребро със същия старши бит? Оказва се, че е вярно. Нека сортираме числата и ги разделим на две групи, спрямо бита на най-старшата позиция  $p$ . За да се построи ребро между двете групи, трябва всички върхове в съответните групи да са свързани, защото всеки XOR на двойка в група е по-малък, отколкото XOR между двете групи. Така рекурсивно може да се реши задачата, използвайки и помощната задача (обаче за минимален XOR).

**Задача 3: (min-xor)** Задачата се състои от едно интересно наблюдение, и то е факта, че минималният XOR на двойка в множеството се състои от съседни числа по големина. Имайки предвид този факт, е достатъчно да се пази `std::multiset` за числата и XOR-овете на съседните по големина елементи на множеството.

### 4 Интересни задачи

**Задача 4: (xoranges)** В задачата виждаме, че всяка от стойностите в даден интервал се повтаря много пъти. Всъщност ни интересува единствено и само четността на броя срещания на дадена стойност  $a_i$  в израза, а именно  $(i - l + 1)(r - i + 1)$ . Оттук, ако  $l$  и  $r$  са с различна четност, веднага следва, че отговорът на заявката е 0 (всеки член се среща четен брой пъти), а ако са с еднаква четност, отговорът на заявката е  $a_l \oplus a_{l+2} \oplus \dots \oplus a_{r-2} \oplus a_r$ ,

което можем да разбираме бързо със сегментно дърво.

**Задача 5: (transfer)** Нека  $N = 255$  и битовете са  $a_1, \dots, a_n$ . Първо, нека се опитаме да поставим нови битове по такъв начин, че да разберем дали се е променил някой бит. Интересно свойство е, че бит се обръща тогава и само тогава, когато XOR-ът се промени, затова, ако добавим бит, равен на  $a_1 \oplus \dots \oplus a_N$ , вторият играч може да разбере кога се е променил бит (точно когато XOR-ът на новите битове е 1). Тази идея сама по себе си не е достатъчно полезна, но бързо става ясно, че можем да не се ограничаваме до цялата редица, а до което и да е нейно подмножество. Така всеки бит от такъв вид, който добавяме, съответства на заявка от вида "има ли променен бит в това множество?" и с  $\log_2(N) = 8$  заявки можем да разберем позицията на променения бит (виж **twins**). Примерни множества, които успяват да разпознаят еднозначно променения бит, са следните: множество  $S_k$  ( $k \in \{0, \dots, 7\}$ ) включва всички индекси на числа, които имат единица като  $k$ -ти бит. Следва да оправим два частни случая:

- Промененият бит е сред новите. За да предотвратим този случай е достатъчно да добавим още един бит, равен на XOR-ът на добавените битове.
- Няма променен бит. С малко замисъл се вижда, че дори и да не знаем този факт и тръгнем да търсим промененият бит, с горепосочените множества процесът ще върне 0. Това означава, че ако индексираме числата от 1, ще работим точно с 256 числа - от 0 до 255.

**Задача 6: (abstract)** Задачата е подобна на 888G, но с побитово или, тоест е логично отново да измислим решение с алгоритъм на Крускал. Ако се пробваме да напишем подобна рекурсия в тази задача, ще получим сложност  $O(N^2)$ , защото текущото множество от върхове не се разделя на непересичащи се множества. Понеже стойностите са малки ( $a_i < 2^B$ ), можем директно да ги обходим - нека в момента разглеждаме всички ребра с тегло  $w$ , тоест искаме да свържем всеки два върха, за които  $a_i \mid a_j = w$ . От това равенство директно следва, че  $a_i$  и  $a_j$  трябва да бъдат подмаски<sup>1</sup> на  $w$ , и обратно - всеки две подмаски на  $w$  ще бъдат свързано заедно най-късно при тегло  $w$ . Използвайки това твърдение, можем ли лесно да свържем всички подмаски на  $w$ ? Оказва се, че сравнително лесно можем да покрием всички подмаски на  $w$ , ако преди това сме обходили всички по-малки тегла. Нека разгледаме случайно  $a_i$ , което е подмаска на  $w$ . Разграничаваме следните два варианта:

- $a_i$  е равно на  $w$ . Тук е достатъчно чрез DSU да обединим числата с останалите намерени накрая.

---

<sup>1</sup>Число  $x$  е подмаска на  $y$ , ако  $x \mid y = y$ , тоест всички единици на  $x$  се срещат и в  $y$ .

- $a_i$  не е равно на  $w$ . С гореспоменатата идея, че броят битове намалява, нека разгледаме числата  $w_1, \dots, w_k$  - преките подмаски<sup>2</sup> на  $w$ . Тогава непременно трябва  $a_i$  да е подмаска на някое  $w_j$  и понеже подмаските на  $w_j$  вече са свързани помежду си, е достатъчно за всяко  $w$  да поддържаме по едно  $a_i$ , което е подмаска на  $w$ , ако съществува такова.

Понеже  $k \leq \log_2(2^B) = B$ , описаното решение е със сложност  $O(NB)$ . Обобщено, алгоритъмът е следният:

- За всяко  $w = 0, \dots, 2^B - 1$  изпълняваме следните стъпки.
- В един списък поставяме позициите на всички срещания на  $w$  в редицата и по един елемент от списъка на всяка пряка подмаска на  $w$ .
- Чрез DSU свързваме всеки два елемента на списъка. За всяко свързваме към общата сума добавяме  $w$ .

## 5 Допълнителни задачи

- xorting
- xorSort
- 2247D2
- xor
- 2180C

---

<sup>2</sup>Число  $x$  е пряка подмаска на  $y$ , ако е подмаска на  $y$  и двете числа се различават в точно един бит.