Переборы с помощью битовых масок

Кориненко Матвей

Март 2022

1 Решение задачи «Сумма K»

Вам уже наверняка приходилось так или иначе сталкиваться с задачей «Сумма K». Кратко, в ней вам дается набор из N чисел, требуется сказать, существует ли подмножество чисел из этого набора, сумма которых равна K.

Вероятнее всего, вы решали эту задачу с использованием рекурсивного перебора. Вы создали функцию, которая «передвигается» по элементам из набора и на каждом из них запускает две ветки рекурсии — в первом случае вы берете элемент в подмножество, во втором вы его пропускаете. Когда вы доходите до последнего элемента, вы проверяете, получилась ли сумма выбранных чисел равной K. Если да — прерываетесь и говорите «YES». А «NO» можно вывести только если после перебора **ВСЕХ** подмножеств вы не нашли ответ, то есть по окончании рекурсии.

2 Представление чисел в компьютере

На время отвлечемся от рассуждений о задачах.

Как представляются числовые данные в компьютере? В двоичной системе счисления с помощью последовательности нулей и единиц.

0	1	1	0	1	0
5	4	3	2	1	0

Например, эта последовательность представляет число 26, так как

$$0 \cdot (2^0) + 1 \cdot (2^1) + 0 \cdot (2^2) + 1 \cdot (2^3) + 1 \cdot (2^4) + 0 \cdot (2^5) = 26$$

Иными словами, в этой последовательности единица на i-ом месте означает наличие в представляемом числе слагаемого 2^i. А можно ли это использовать в своих интересах?

3 Битовые маски

Давайте теперь единица на i-том месте в числе будет отвечать не только за наличие в нем слагаемого 2^i, но и за наличие в каком-либо подмножестве i-го элемента из стороннего набора.

$mask$	0	1	1	0	1	0
$data$	2	1	3	9	4	6

Последовательность битов, обозначенная в примере за $mask$ называется битовой маской.

Битовая маска — определённые данные, которые используются для маскирования — выбора отдельных битов или полей из нескольких битов из двоичной строки или числа.

В нашем случае это можно интерпретировать как из набора элементов $\{2, 1, 3, 9, 4, 6\}$ выбирается подмножество $\{1, 3, 4\}$.

Таким образом есть возможность выбрать подмножество элементов из множества размером вплоть до 64 элементов (*unsigned long long*). Теперь разберемся, как это можно реализовать.

4 Реализация и применение битовых масок

Представим, что у нас имеется набор из n (пусть $n \leq 10$) элементов и нам зачем-то нужно перебрать все его подмножества. Поэтому битовая маска должна иметь n битов, значение каждого из которых будет отвечать за наличие соответствующего элемента в подмножестве.

Поскольку $n \leq 10$, в качестве числового типа данных для маски можно взять *int*, так как в него влезает число вплоть до 2^{31}.

Теперь нам нужно понять, в какой момент перебор элементов нужно закончить. Если рассматривать маску именно как число, то становится понятно, что перебирать значения нужно в пределах полуинтервала $[0, 2^n)$. Число 0 отвечает за пустое подмножество и является самым маленьким числом, которое можно получить с использованием n битов. А $2^n - 1$ — максимальное число, представимое с помощью тех же n битов, и отвечающее за подмножество, включающее в себя все n элементов набора.

Соответственно, перебор всех масок размера n можно удобно осуществлять путем перебора всех чисел от 0 до $2^n - 1$ с помощью простого цикла.

```
for (int mask = 0; mask < (1 << n); mask++) {
    // ?
}
```

Остался вопрос — как проверять наличие определенного элемента в подмножестве $mask$? Для этого опять стоит вспомнить, что битовая маска представляет из себя просто число, а потому с ней допустимы привычные для чисел операции, в том числе и битовые.

Давайте для проверки наличия i-го элемента в подмножестве создадим новое число, которое будет иметь единичный бит только на i-ом месте. Обозначим его за x. Саму проверку будем осуществлять с помощью операции AND: если применить ее к числам $mask$ и x, то мы получим значение большее нуля только при условии, что в $mask$ на i-ом месте стоит единичный бит.

Дополним наш код.

```
for (int mask = 0; mask < (1 << n); mask++) {
    for (int i = 0; i < n; i++) {
        if (mask & (1 << i)) {
            // что-то делаем с этим элементом
        }
    }
}
```

Таким образом, код для решения задачи «Сумма K» может выглядеть достаточно просто и коротко:

```
for (int mask = 0; mask < (1 << n); mask++) {
    ll cur_sum = 0;
    for (int i = 0; i < n; i++) {
        if (mask & (1 << i)) {
            cur_sum += a[i];
        }
    }
    if (cur_sum == k) {
        cout << "YES";
        return 0;
    }
}
```

Среди преимуществ перебора с использованием битовых масок есть и весьма очевидные — меньшее использование ресурсов компьютера (у итеративного решения стек рекурсивных вызовов пуст), малый объем кода (если понимать идею масок).

А есть достаточно неочевидный — представление множеств в виде чисел позволяет решать некоторые задачи динамического программирования. Но об этом мы поговорим на следующих лекциях.

5 Почему важно уметь писать переборные решения

На большинстве проводимых школьных олимпиад каждая задача разбивается на подзадачи для возможности получения частичных баллов. Поэтому, во-первых, в случае отсутствия идеи для решения задачи на полный балл, написать решение получающее частичный будет лучше, чем получить за нее 0.

Вторая причина, вероятно, более весомая. Используя переборное решение можно найти ошибку в оптимальном при помощи стресс-теста. Да, его не для любой задачи написать можно легко и быстро, но пренебрегать такой возможностью не стоит.

В-третьих, для решения определенных задач достаточно написать переборное решение с отсечением. При переборе некоторых состояний, можно сразу понять, что в дальнейшем они вам точно не пригодятся, а значит, данную ветку перебора можно обрывать и переходить к следующей.