Разберём следующую задачу, которая часто встречается как подзадача.

> Дано множество из n элементов a_1, a_2, $\ldots$, a_n. Нужно для каждого из его подмножеств посчитать сумму элементов и уметь ее быстро возвращать.

Иначе говоря, нужно заполнить массив или вектор sum, где sum[mask] — сумма элементов в подмножестве, задаваемом маской mask. Вместо суммы в этой задаче может быть произведение, минимум или какая-нибудь другая функция.

На основе предыдущей лекции мы можем написать следующее решение (см. рис. 1).

```
for (int mask = 0; mask < (1 << n); mask++)
{
    sum[mask] = 0;
    for (int i = 0; i < n; i++)
        if (mask & (1 << i))
            sum[mask] += a[i];
}
```

Рис. 1. Решение перебором

Мы перебираем маски от 0 до 2^n:

for (**int** mask = 0; mask < (1 << n); mask++)

Пусть мы фиксировали какую-то маску. Сумма для неё сначала присваивается нулю:

sum[mask] = 0;

Дальше мы перебираем биты от 0 до $n-1$:

for (**int** i = 0; i < n; i++)

Проверяем, есть ли в маске i-й бит:

if (mask & (1 << i))

Если да (в i-м бите стоит единица) — прибавляем соответствующее число a[i] к текущей сумме:

sum[mask] += a[i];

Оценим время работы этого решения. Внешний цикл выполняет 2^n итераций, внутренний — n. Получаем $O(n2^n)$ действий.

Можно ли как-то ускорить это решение? Ответ в названии лекции — динамика. Нужно каким-то образом научиться вычислять следующие значения сумм через предыдущие.

Например, можно поступить так (см. рис. 2). Как и в предыдущем решении, в цикле по i мы перебираем биты от 0 до $n-1$ и находим бит, который содержится в маске. Если мы отбросим этот бит, то получим другую маску, в которой меньше битов. Как целое число, она идет раньше нашей маски, а значит, сумма для нее уже посчитана, и мы можем просто прибавить к ней a[i]. Маска без i-го бита будет получаться как (mask ^ (1 $<<$ i)). Мы знаем, что в mask i-й бит равен 1, поэтому операция XOR с числом, содержащим одну единицу в i-м бите, приведёт к обнулению этого бита. После этого значение sum[mask] посчитано и можно сделать break — выход из внутреннего цикла.

```
for (int mask = 0; mask < (1 << n); mask++)
{
    for (int i = 0; i < n; i++)
        if (mask & (1 << i))
        {
            sum[mask] = sum[mask ^ (1 << i)] + a[i];
            break;
        }
}
```

Рис. 2. Решение динамикой

Решение очень похоже не предыдущее, но предыдущее решение было перебором, а это — динамика. Оценим время её работы. С первого взгляда, все так же. Внешний цикл выполняет 2^n итераций, внутренний — n итераций. Но оказывается, что break существенно ускоряет работу программы. Первый бит, в котором стоит единица, в большинстве случаев будет находиться очень быстро. В половине случаев — уже на первом шаге, потому что половина чисел — нечётные. В половине из оставшихся случаев — на втором шаге и т.д. Это означает, что первый шаг внутреннего цикла будет сделан для всех масок, второй — только для половины, третий — для четверти, и т.д. Получаем следующую сумму:

$$2^n\left(1+\frac{1}{2}+\frac{1}{4}+\cdots+\frac{1}{2^{n-1}}\right). \tag{1}$$

Множитель 2^n — это общее количество масок. Очевидно, что (1) — сумма убывающей геометрической прогрессии. Она не превосходит 2^{n+1}.

В итоге получаем оценку времени работы $O(2^n)$. Это меньше, чем время работы переборного решения $O(n2^n)$. Отличие, конечно, не очень большое с учётом того, что такие решения работают за приемлемое время примерно до $n = 25$.

Эта задача приводится с двумя целями. Во-первых, показать, что битовые маски можно использовать как состояния в динамическом программировании. Эта идея очень часто встречается в олимпиадных задачах. Во-вторых, мы на примере разобрали, как считать асимптотику времени работы решений, основанных на битовых масках.