Модуль 3 Динамическое программирование
Лекция 3.1 Задача о замощении полосы доминошками

На этой лекции мы начнём знакомство с методом динамического программирования, который часто применяется при решении олимпиадных задач. Рассмотрим задачу.

Задача о замощении полосы доминошками
Имеется полоса размера $2 \times n$. Нужно найти число способов замостить ее доминошками размера 1×2. Доминошки не должны пересекаться и выходить за пределы полосы.

На рис. 1 показан пример такого замощения для $n = 8$.

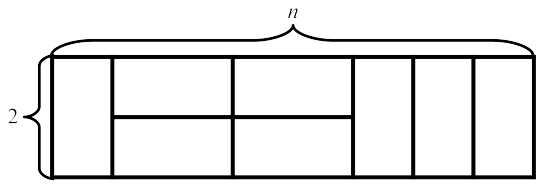

Рис. 1. Пример для $n = 8$

Для $n = 4$ будет 5 различных замощений (см. рис. 2), поэтому ответ на задачу равен 5.

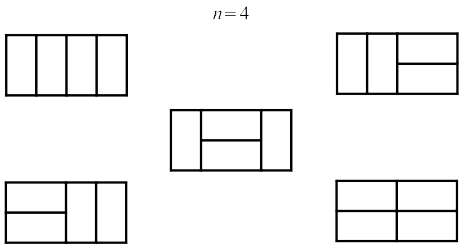

Рис. 2. Пример для $n = 4$

Сначала разберём одно не очень эффективное решение задачи. Нетрудно заметить, что у нас есть вертикальные доминошки и есть горизонтальные, которые идут парами и занимают квадраты 2×2. Каждое замощение соответствует разбиению числа n на слагаемые, каждое из которых 1 или 2. Например, замощению на рис. 1 соответствует разбиение $1 + 2 + 2 + 1 + 1 + 1$.

Порядок слагаемых важен — разным порядкам соответствуют разные разбиения. Посчитать количество разбиений числа на слагаемые можно при помощи рекурсивного перебора.

Функция rec на рис. 3 перебирает все возможные разбиения. Ей передаются два параметра: idx — индекс в векторе a, в который помещаются слагаемые, и sum — накопленная сумма. На каждом уровне рекурсии функция пытается добавлять в вектор a слагаемое 1 или 2. Ответ подсчитывается в глобальной переменной ans. Когда мы находим новый способ — её значение увеличивается на 1. Заметим, что вектор a у нас фактически не используется для подсчета ответа. Нам нужно количество разбиений, а не сами разбиения. Поэтому вектор a и переменную idx можно просто убрать (см. рис. 4).

```
int ans = 0;

void rec(int idx, int sum)
{
    if (sum > n)
        return;
    if (sum == n)
    {
        ans++;
        return;
    }

    a[idx] = 1;
    rec(idx + 1, sum + 1);
    a[idx] = 2;
    rec(idx + 1, sum + 2);
}
```

Рис. 3

```
int ans = 0;

void rec(int sum)
{
    if (sum > n)
        return;
    if (sum == n)
    {
        ans++;
        return;
    }

    rec(sum + 1);
    rec(sum + 2);
}
```

Рис. 4

Оценим время работы функции на рис. 4. По рекурсивным функциям обычно не очевидно, сколько действий они выполняют. Будем отталкиваться от то-

го, что для функции rec число действий будет не меньше полученного в итоге ответа ans. Ответ будет получаться довольно большим. Например, для $n = 40$ программа выводит 165580141, а для $n = 50$ уже не успевает отработать за 10 секунд. Таким образом, ответ растёт очень быстро с ростом n.

Чтобы решить задачу при больших n, нужно каким-то образом посчитать разбиения на слагаемые без перебора самих разбиений. Пусть у нас имеется какое-нибудь разбиение. Последнее число в этой разбиении может быть либо 1, либо 2. Если мы отбросим последнюю единицу, то получим некоторое разбиение на единицы и двойки числа $(n - 1)$. Если же последняя двойка, и мы ее отбросим — получим разбиение числа $(n - 2)$ (см. рис. 5).

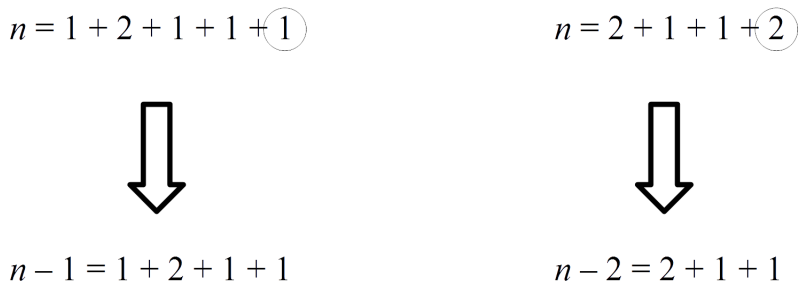

Рис. 5

Обозначим ответ на задачу для числа n за d_n. Разбиения числа n на слагаемые с последней единицей соответствуют способам разбиения числа $(n - 1)$, количество которых равно d_{n-1}, а разбиения числа n с последней двойкой соответствуют разбиениям $(n - 2)$, количество которых — d_{n-2}. Получаем, что общее число разбиений числа n равно сумме этих количеств, то есть

$$d_n = d_{n-1} + d_{n-2}. \tag{1}$$

Мы получили формулу, по которой можно посчитать ответ для заданного n. Нужно ещё задать начальные значения. Нам известно, что $d_1 = 1$: существует только одно замощение полосы 2×1 одной доминошкой, и $d_2 = 2$: полосу 2×2 можно замостить двумя горизонтальными или двумя вертикальными доминошками, всего два способа. После этого можно заполнить таблицу на рис. 6 по формуле (1). Значения в таблице — это ответы на задачу при разных значениях n. Каждое следующее число получается как сумма двух предыдущих: $3 = 1 + 2$, $5 = 2 + 3$ и т.д.

n	1	2	3	4	5	6	7	8	9	10
d_n	1	2	3	5	8	13	21	34	55	89

Рис. 6

Упражнение 1.3.1
Дана полоса размера $3 \times n$. Найдите число способов замостить её триминошками 1×3 при $n = 10$.

На рис. 7 приведён код решения задачи с доминошками.

```
vector<long long> d(n + 1);

d[0] = d[1] = 1;
for (int i = 2; i <= n; i++)
    d[i] = d[i - 1] + d[i - 2];
```

Рис. 7

Создадим вектор ответов d:

vector<**long long**> d(n + 1);

Инициализируем первые два его значения:

d[0] = d[1] = 1;

Здесь мы искусственно добавляем задачу для $n = 0$, то есть для пустой полосы. Эту полосу можно замостить одним способом — нулём доминошек. Заметим, что это значение не нарушает общую закономерность: d[0] + d[1] = 2, что равно d[2], то есть ответу для $n = 2$. Далее идет подсчёт в цикле по формуле (1):

for (**int** i = 2; i <= n; i++)
    d[i] = d[i - 1] + d[i - 2];

Программа получилась даже проще, чем перебор, и выполняет всего $O(n)$ действий.

Этот метод решения называется *динамическим программированием*, или *динамикой*. В чём состоит его основная идея? Мы решаем не одну задачу для заданного n, а сразу много задач для всех предыдущих значений. Решения каждой задачи получается через решения меньших задач по рекуррентной формуле (1). Обязательный этап — инициализация, присвоение начальных значений. Здесь мы присваиваем два начальных значения — d[0] и d[1], потому что в рекуррентной формуле используются два предыдущих значения — d[i-1] и d[i-2]. Инициализации только одного начального значения будет недостаточно.

Таким образом, динамика состоит из инициализации начальных значений и вычислений по рекуррентной формуле, которая еще называется переходом динамики. На следующих лекциях мы разберём другие задачи на динамическое программирование.

Отметим, что полученная последовательность чисел:

$$1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, \ldots,$$

в которой каждое следующее число равно сумме двух предыдущих, называется последовательностью *чисел Фибоначчи* и часто встречается в олимпиадных задачах.