На этой лекции мы вернёмся к задаче о размене.

Задача о размене
Имеются монеты n номиналов $a_1, a_2, \ldots, a_n$ и сумма s. Нужно набрать эту сумму наименьшим количеством монет. Можно использовать любое число монет каждого номинала.

Мы уже встречались с этой задачей на лекции про жадные алгоритмы и выяснили, что жадное решение для неё работает не при любых номиналах монет. Подумаем, как решить эту задачу при помощи динамического программирования.

Обозначим через d_i ответ на задачу для суммы i, то есть минимальное число монет, необходимое, чтобы набрать сумму i. Выведем формулу для вычисления d_i. Переберём номинал последней монеты, которая будет участвовать в размене суммы i, пусть она имеет номинал a_j. Тогда вопрос о том, какое минимальное число монет нужно, чтобы набрать сумму i, сводится к вопросу о том, какое минимальное число монет нужно, чтобы набрать сумму $i - a_j$. Мы свели задачу к меньшим аналогичным задачам. Из ответов для меньших задач нужно выбрать минимум, поскольку мы ищем оптимальное решение, и прибавить единицу, потому что мы должны добавить j-ю монету. Мы получили следующую формулу для перехода динамики, на основе которой можно писать программу.

$$d_i = \min_{1 \leqslant j \leqslant n} d_{i-a_j} + 1. \tag{1}$$

Разберём реализацию решения (см. рис. 1).

```
vector<int> d(s + 1);

d[0] = 0;
for (int i = 1; i <= s; i++)
{
    d[i] = INF;
    for (int j = 0; j < n; j++)
        if (i - a[j] >= 0)
            d[i] = min(d[i], d[i - a[j]] + 1);
}
cout << d[s] << endl;
```

Рис. 1. Решение задачи о размене

Объявим вектор d для ответов. Нам понадобятся его элементы до номера s включительно, поэтому делаем ему размер $(s + 1)$.

```
vector<int> d(s + 1);
```

Сначала инициализируем нулевое значение вектора нулём, потому что нулевую сумму можно набрать нулевым количеством монет.

```
d[0] = 0;
```

Затем перебираем в цикле значения i от 1 до заданной суммы s.

```
for (int i = 1; i <= s; i++)
```

Пусть мы хотим решить задачу для суммы i и уже решили для всех предыдущих значений. По формуле (1) нам нужно будет выбрать минимум из нескольких чисел, поэтому инициализируем d[i] очень большой константой INF, так называемой «бесконечностью».

```
d[i] = INF;
```

«Бесконечность» должна быть заведомо больше ответа. Например, можно взять в качестве INF число $(s + 1)$.

Во внутреннем цикле перебираем монеты. Считаем, что при реализации они нумеруются с нуля.

```
for (int j = 0; j < n; j++)
```

Затем идёт помещение в d[i] нового значения согласно формуле (1), если это значение меньше.

```
d[i] = min(d[i], d[i - a[j]] + 1);
```

Чтобы индекс (i - a[j]) не оказался отрицательным, предварительно выполняется проверка.

```
if (i - a[j] >= 0)
```

В результате элемент d[s] будет равен ответу для суммы s.

Кстати, может получиться, что для какой-то суммы значение d[i] так и не поменяется, останется равным INF. Это означает, что сумму i вообще нельзя набрать заданными монетами. Например, если она меньше номиналов всех монет.

Оценим время работы программы. Здесь всё очевидно: программа имеет два вложенных цикла, один до s, другой до n. Получаем $O(ns)$ действий. Это решение будет работать за приемлемое время, например, при следующих ограничениях: $n \leqslant 100$, $s \leqslant 10^6$. При совсем больших s (например, до 10^9) программа работать не будет, потому что мы не сможем создать массив такого большого размера.

Итак, мы получили ответ. Обсудим, как вывести сертификат. Сертификатом в данном случае будет сам набор из минимального числа монет с заданной суммой. Например, сертификат может выглядеть следующим образом: $10 + 10 + 2 + 5 + 10 + 2$. Здесь используются монеты номиналов 10, 2 и 5. Определённый порядок не требуется, хотя если и потребуется, мы сможем применить сортировку.

На рис. 2 отмечены изменения в коде для вывода сертификата.

```
vector<int> d(s + 1);
vector<int> p(s + 1);

d[0] = 0;
for (int i = 1; i <= s; i++)
{
    d[i] = INF;
    for (int j = 0; j < n; j++)
        if (i - a[j] >= 0 && d[i - a[j]] + 1 < d[i])
        {
            d[i] = d[i - a[j]] + 1;
            p[i] = a[j];
        }
}
cout << d[s] << endl;
recout(s);
```

Рис. 2. Вывод сертификата

Для каждой суммы i вместе с минимальным количеством монет d[i] определяется номинал монеты, которую мы берём последней при оптимальном размене. Для этого создаётся вектор p.

vector<**int**> p(s + 1);

Далее мы перебираем номиналы монет в цикле. Проверяем, выгодно ли нам брать последней монету с номиналом a[j]:

if (i - a[j] >= 0 && d[i - a[j]] + 1 < d[i])

Пересчитываем значение d[i].

d[i] = d[i - a[j]] + 1;

И помещаем a[j] в вектор p.

p[i] = a[j];

В конце мы вызываем рекурсивную функцию recout от s, которая выведет сертификат.

recout(s);

Реализация функции recout представлена на рис. 3.

Функции передаётся сумма i, которую нужно набрать. Если $i = 0$, то никакие монеты нам не нужны. Выходим из функции.

if (i == 0)
    return;

В противном случае мы вычитаем из i последнюю монету, которую нужно взять при оптимальном размене (она запомнена в ячейке p[i]), и делаем рекурсивный вывод ответа от оставшейся суммы.

```
void recout(int i)
{
    if (i == 0)
        return;
    recout(i - p[i]);
    if (i - p[i] > 0)
        cout << "+";
    cout << p[i];
}
```

Рис. 3. Вывод сертификата

recout(i - p[i]);

Далее мы добавляем к ответу знак плюс, если наше слагаемое — не самое первое:

```
if (i - p[i] > 0)
    cout << "+" << endl;
```

И выводим слагаемое p[i].

```
cout << p[i];
```

В результате мы вызовем функцию от заданной суммы s, найдём оптимальное последнее слагаемое, вычтем его, вызовем функцию от новой суммы, уже меньшей, вычтем следующее слагаемое и т.д., пока не дойдём до нуля. На нуле выполнится вход из рекурсии, и дальше мы постепенно выведем все слагаемые. Здесь применяется такая же идея, как в задаче про жучка для вывода его пути: сохранять для каждого состояния информацию о том, как мы в него пришли, и потом рекурсивно выводить сертификат.

Таким образом, мы разобрали решение задачи о размене методом динамического программирования и научились строить сертификат.