Жадные алгоритмы

Жадными называют класс алгоритмов, заключающихся в принятии локально оптимальных решений на каждом этапе. Так как локально оптимальное решение вычислить гораздо проще, чем глобально оптимальное, такие алгоритмы обычно имеют хорошую асимптотику.

В некоторых случаях жадные алгоритмы приводят к оптимальным конечным решениям, а в других — нет. Придумать и особенно доказать корректность жадных алгоритмов часто бывает очень сложно.

В этой статье мы разберем несколько классических примеров задач, решающихся жадным алгоритмом.

Монетки

Задача. Монетная система некоторого государства состоит из монет достоинством $a_1 = 1 < a_2 < a_3 < \ldots < a_n$, причём каждый следующий номинал делится на предыдущий. Требуется выдать сумму S наименьшим возможным количеством монет.

Жадный алгоритм решения этой задачи заключается в том, что нужно сначала взять наибольшее количество монет достоинства a_n (а именно, $\lfloor \frac{S}{a_n} \rfloor$), затем для набора остатка взять наибольшее количество монет достоинства a_{n-1} и так далее. Так как $a_1 = 1$, в конце мы всегда наберем нужную сумму.

```
a = [1, 2, 5, 10, 50, 100, 500, 1000, 2000, 5000]

def solution(s):
    coins = []
    for x in reversed(a):
        coins += [x] * (s // x)
        s %= x
    return coins
```

Заметим, что без предположения, что номиналы монет делятся друг на друга, алгоритм не всегда выводит оптимальное решение, и в этом случае задачу остается решать динамическим программированием. Например, сумму в 24 рубля монетами в 1, 5 и 7 рублей жадный алгоритм разменивает как $(7 \times 3 + 1 \times 3)$, в то время как оптимальным будет $(7 \times 2 + 5 \times 2)$.

Корректность. Докажем корректность жадного алгоритма от противного: пусть максимальная монета имеет достоинство $a_n < S$, но в оптимальном ответе её нет.

Пусть в оптимальном ответе монета с номиналом a_i встретилась b_i раз:

$$S = \sum a_i \cdot b_i$$

Если $b_i \geq \frac{a_{i+1}}{a_i}$, то $\frac{a_{i+1}}{a_i}$ монет можно заменить на одну монету достоинства a_{i+1}, а значит это не оптимальный ответ. Следовательно, $b_i \leq \frac{a_{i+1}}{a_i} - 1$.

Теперь посчитаем, какой может быть максимальная сумма всех достоинств всех монет в оптимальном ответе, если мы не брали максимальные монеты:

$$\begin{aligned} S &= a_1 \cdot b_1 + a_2 \cdot b_2 + \ldots + a_{n-1} \cdot b_{n-1} \\ &\leq a_1 \cdot (\frac{a_2}{a_1} - 1) + a_2 \cdot (\frac{a_3}{a_2} - 1) + \ldots + a_{n-1} \cdot (\frac{a_n}{a_{n-1}} - 1) \\ &= (a_2 - a_1) + (a_3 - a_2) + \ldots + (a_n - a_{n-1}) \\ &= a_n - a_1 < a_n \end{aligned}$$

Отсюда делаем вывод, что если $S \geq a_n$, то в оптимальный ответ всегда придется взять максимальную монету размера a_k, потому что все меньшие монеты просто не смогут оптимальном ответе давать так много. Аналогичным размышлением про наборы из меньших монет получаем такой же результат для всех $S < a_n$, откуда следует корректность жадного алгоритма.

Также этот алгоритм работает не только для делящихся друг на друга номинациях, но и для некоторых других. Такие монетные системы, где жадный алгоритм работает, называют *каноническими*.

Упражнение. Верно ли, что алгоритм работает корректно для российской монетной системы (1, 2, 5…)?

Рюкзак с делимыми предметами

Задача. Пусть есть рюкзак с вместимостью не более, чем W грамм и n предметов весом w_i грамм и стоимостью c_i за грамм. Мы умеем отрезать от любого предмета целое количество грамм. Требуется набрать рюкзак максимальной стоимости.

Отсортируем предметы по убыванию «плотности ценности» $\frac{c_i}{w_i}$ и будем брать их жадно в таком порядке. От последнего предмета, который не влезет полностью, возьмем часть.

Корректность. Мысленно разделим все предметы на w_i кусочков по 1 грамм, и при этом их ценность стала равна $\frac{c_i}{w_i}$. Понятно, что из кусочков одинакового веса 1 грамм всегда оптимально просто взять кусочки с максимальной ценностью, что мы фактически и делаем в нашем алгоритме.

Итоговая асимптотика $O(n \log n)$ на сортировку.

Выбор заявок

Задача. Даны заявки на проведение занятий в некоторой аудитории. В каждой заявке указаны начало l_i и конец r_i занятия. Нужно из всех заявок оставить как можно больше таким образом, чтобы они не пересекались.

Здесь жадность становится не такой уже очевидной, потому что неясно, в каком порядке рассматривать заявки и как их «жадно» набирать.

Посмотрим на *первую по времени конца* заявку. Заметим, что нам всегда выгодно включить её в оптимальный ответ — она заканчивается раньше всех остальных, а поэтому если в оптимальном ответе первая заявка какая-то другая, мы можем безболезненно заменить её на первую по времени конца, и при этом новых пересечений не появится, так как мы просто сдвинули самую первую заявку еще левее.

После того, как мы выбрали первую по времени конца, уберем из рассмотрения все, которые с ней пересекаются, и так же выберем из оставшихся первую по времени конца, снова уберем все пересекающиеся и так далее, пока заявки не кончатся.

При реализации удобно не удалять отрезки явно, а просто отсортировать отрезки по времени конца и поддерживать время, когда кончается последнее взятое занятие.

```
struct activity {
    int l, r;
};

int solve(vector<activity> activities) {
    sort(activities.begin(), activities.end(), [](activity a, activity b) {
        return a.r < b.r;
    });
    int cnt = 0, last = 0;
    for (activity a : activities)
        if (a.l >= last)
            last = a.r, cnt++;
    return cnt;
}
```

Асимптотика $O(n \log n)$ на сортировку.