Discrete resource partition problems

Дмитрий Иващенко

October 2015

1 Задача

Пусть имеется число $n \in \mathbb{N}$, а также m неубывающих дискретных функций $f_1, \ldots, f_m \colon [0; N] \to \mathbb{T}$, где $\mathbb{T}$ — это произвольное множество с линейным порядком. Необходимо найти разбиение числа n на слагаемые $n_1, \ldots, n_m$ так, чтобы $\min\limits_{1 \leqslant i \leqslant m} f_i(n_i) \to \max$. Задачу можно интерпретировать как вложение некоторого дискретного ресурса, например, в регионы страны с целью максимизировать благосостояние наименее развитого.

В случае $\mathbb{T} \subset \mathbb{N}$ можно решить задачу за время $O(m \log C \log n)$, где $C = \max\limits_{1 \leqslant i \leqslant m, 1 \leqslant j \leqslant i} f_i(j)$. C можно считать константой, так как основные операции с целыми числами имеют временную сложность $O(\log C)$, которую привычнее считать константой.

Этот алгоритм подразумевает бинарный поиск по реальному значению ответа, что может быть затруднительно в случае нетривиального порядка на $\mathbb{T}$. В частности реализация, использующая бинарный поиск по ответу, подразумевает быстрое нахождение элемента, разбивающего пространство поиска на равные части, что не всегда бывает простой задачей. Естественные примеры такого рода включают, в частности, кортежи из нескольких элементов или множества с более экзотической структурой, такие как множества рациональных чисел со знаменателем не превосходящим некотрого d, а также некоторые другие.

Описанный ниже алгоритм позволяет безотносительно природы множества $\mathbb{T}$ решать описанную задачу за время $O(m \log^2 n)$, то есть линейно по m при фиксированном n. Единственное предположение о природе входных данных это возможность вычислять значения функций в точках и сравнивать их за $O(1)$. Также для простоты, предположим, что среди значений функций нет равных, для этого можно ввести на равных элементах порядок произвольным образом.

2 Алгоритм Блума-Флойда-Пратта-Риверста-Тарьяна

Для начала напомним, как можно находить порядковую статистику в произвольном массиве из m элементов за линейное время в худшем случае. Для этого можно поступить следующим образом:

- Разбить массив на $\left[\frac{m}{5}\right]$ равных (кроме, возможно, последней) частей и отсортировать каждую каким-нибудь алгоритмом.
- Из медиан каждой из частей выбрать медианное значение X рекурсивно.
- Разбить все элементы на две группы: S_1, содержащую все элементы, меньшие X, и S_2, содержащую все остальные.
- Если S_1 содержит порядковую статистику (это можно узнать, сравнив его размер и статистику, которую мы ищем), то перейдем рекурсивно в него. В противном случае нужно перейти рекурсивно ко множеству S_2.

Легко видеть, что как минимум четверть всех элементов будет меньше элемента X, и не менее четверти всех элементов будет больше X. Тогда время работы алгоритма описывется рекуррентой $T(m) = T(\frac{m}{5}) + T(\frac{3m}{4}) + O(m)$, которая имеет решение $T(m) = O(m)$.

Стоит отметить, что константу 5 уменьшить нельзя, так как $\frac{1}{3} + \frac{3}{4} > 1$ и рекуррента перестанет быть линейной.

3 Алгоритм за $O(m \log^2 nm)$

Основная идея состоит в применении идеи выбора «медианы медиан» для замены бинарного поиска по ответу. Будем для каждой функции поддерживать числа L_i, R_i, в начале равные 0 и N соответственно. Будем поддерживать инвариант, состоящий в том, что для всех $1 \leqslant i \leqslant m \to f_i(L_i) \leqslant ans \leqslant f_i(R_i)$, где ans — это искомое значение ответа. Обозначим через Len_i величину $R_i - L_i + 1$, а также $S = \sum\limits_{1 \leqslant i \leqslant m} Len_i$.

Ясно, что запрос «верно ли что ответ $> X$?» для некоторого значения X можно реализовать за время $O(m \log n)$, если найти для каждой функции f_i минимальное значение $j : L_i \leqslant j \leqslant R_i, f_i(j) > X$ и проверить, что сумма таких значений по всем столбцам не превосходит n.

Каждая фаза алгоритма будет заключаться в нахождении подходящего элемента X и запросе, описанном выше и пересчёте значений L_i, R_i.

Для выбора подходящего X выберем у каждой функции значение $X_i = f_i(\left[\frac{L_i+R_i}{2}\right])$ и сформируем вспомогательный массив пар (X_i, Len_i). Этот массив отсортируем по первой координате и выберем первый такой элемент (X_k, Len_k), что $\sum\limits_{i: X_i \leqslant X_k} Len_i \geqslant \left[\frac{S}{2}\right]$. Если выбрать X_k в качестве X, мы гарантируем, что примерно (с точностью до $O(n)$) четверть всех элементов будет больше X и примерно четверть будет меньше. Чтобы избавиться от возможной ошибки в $O(n)$, которая может стать существенной, если S достаточно мало, можно сделать $O(1)$ запросов на сравнение ответа со значениями функции f_k чтобы можно было исправить L_k, R_k, чтобы величина Len_k уменьшилась так, чтобы нужная сумма была равна в точности $\left[\frac{S}{2}\right]$.

После выбора подходящего элемента X мы сделаем запрос на сравнение ответа с ним и пересчитаем лишь некоторые L_i, R_i. Если ответ больше X, то нужно всем функциям, для которых $X_i \leqslant X_k$ изменить L_i на $\left[\frac{L_i+R_i}{2}\right]$. Иначе нужно для всех $i : X_i > X_k$ изменить R_i на ту же величину.

Выбором X мы гарантируем, что величина S уменьшиться как минимум на четверть, поэтому фаз алгоритма будет $O(\log nm)$. На каждой итерации самые затратные действия — сортировка и запрос на проверку ответа. Их общая временная сложность составляет $O(m \log n + m \log m) = O(m \log nm)$. Таким образом, суммарное время работы алгоритма составляет $O(m \log^2 nm)$.

4 Улучшение до $O(m \log nm \log n)$

Чтобы немного ослабить зависимость от m можно прибегнуть к той же самой идее. Долгой и, вероятно, избыточной частью алгоритма является сортировка массива пар (X_i, Len_i) для нахождения подходящего значения k. Однако здесь снова можно применить метод выбора «медианы медиан», чтобы достигнуть линейной асимптотики.

Итак, поймем, что модфицированный алгоритм Блума-Флойда-Пратта-Риверста-Тарьяна применим и в этом случае. Для этого нужно снова разбить массив на части по 5 элементов и выбрать «медиану медиан» X. Не изменится и разбиение на группы S_1 и S_2. Однако процедура определения принадлежности искомого значения k одному из множеств слегка изменится: нужно переходить в множество S_1, если сумма величины Len_i в нём превосходит искомую отметку (будем передавать её в рекурсию, изначально она равна $\left[\frac{S}{2}\right]$), иначе нужно переходить в S_2.

Так как по-прежнему на каждом шаге отделяется хотя бы четверть элементов, то время работы алгоитма составляет $O(m)$, а каждая фаза алгоритма решения исходной задачи ускоряется до $O(m \log n)$, что приводит к общей асимптотике $O(m \log nm \log n)$.

5 Улучшение до $O(m \log^2 n)$

Чтобы уменьшить число итераций рекурсии, отметим, что бинарный поиск по ответу становится неэффективным по мере уменьшения S. Поэтому поступим следующим образом: запустим $O(logn)$ итераций алгоритма так, чтобы значение S уменьшилось до $O(m)$. После этого продолжим те же итерации, но запрос на сравнение ответа с числом будем выполнять не за $O(m \log n)$, а простым линейным проходом за $O(S)$ в сумме. Тогда время работы второй части будет описываться рекуррентой $T(S) = T(\frac{3S}{4}) + O(S)$, которая разрешается как $O(S)$. Таким образом суммарное время работы алгоритма составит $O(m \log^2 n) + O(m) = O(m \log^2 n)$.

Стоит отметить, что оптимально выбирая пропорцию числа итераций первого вида, можно добиться оценки в $O\left(m \log n \log\left(\frac{n}{\log n}\right)\right)$, однако улучшение получается небольшим.