First to Solve

Идея: Геннадий Короткевич
Разработка: Геннадий Короткевич

Разделим задачу на две части.

В первой части заполним матрицу $p(i, j, t)$ — вероятность того, что участник i решит задачу j ровно через t минут после начала соревнования.

Пусть список задач, которые участник i умеет решать, выглядит как $j_1, j_2, \ldots, j_s$. Зафиксируем конкретную задачу j из этого списка. Переберём, какой по счёту она окажется в списке участника i после случайной перестановки — обозначим эту позицию c. Рассмотрим также конкретный момент времени t.

В скольких перестановках задач $j_1, j_2, \ldots, j_s$ задача j окажется на c-м месте, и при этом окажется решена на минуте t?

Пусть $f(i, j, t, c)$ — число подмножеств задач $j_1, j_2, \ldots, j_s$ размера $c - 1$, не содержащих j, с суммой соответствующих элементов $a_{i,*}$ равной $t - a_{i,j}$. Можно заметить, что ответ на вопрос выше равен $f(i, j, t, c) \cdot (c - 1)! \cdot (s - c)!$.

Суммируя все количества перестановок по различным c и деля на общее число перестановок, чтобы перейти к вероятностям, получим формулу для $p(i, j, t)$:

$$p(i, j, t) = \sum_{c=1}^{s} f(i, j, t, c) \cdot (c - 1)! \cdot (s - c)!/s!.$$

Осталось научиться эффективно вычислять $f(i, j, t, c)$. Заметим, что перед нами вариация задачи о рюкзаке, которую можно решать с помощью динамического программирования. Кроме того, при фиксированном i для разных j задачи практически эквивалентны, и отличаются между собой только отсутствием одного предмета.

Для фиксированного i вычислим $g(c, t)$ — число подмножеств $a_{i,j_1}, \ldots, a_{i,j_s}$ размера c с суммой t — с помощью динамического программирования за $O(m^2k)$, добавляя все m «предметов» по одному, каждый за $O(mk)$. Теперь для каждого j давайте «откатим» добавление предмета $a_{i,j}$ — проделаем те же операции, что и при добавлении элемента, но в обратном порядке и с противоположными знаками. В этот момент мы получим матрицу с суммами подмножеств как раз всех элементов, кроме j-го, что и требовалось. Потом вернём предмет $a_{i,j}$ обратно. Так как удаление одного предмета работает за такое же время, как и добавление, все j будут обработаны в сумме за $O(m^2k)$.

Вспоминая, что мы фиксировали i, итоговое время работы всей первой части составит $O(nm^2k)$. При аккуратной реализации проходили по времени и менее эффективные алгоритмы — например, с асимптотиками $O(nm^2k \log m)$ и $O(nm^3k)$.

Во второй части задачи, используя матрицу $p(i, j, t)$, вычислим искомые ответы.

Зафиксируем задачу j и начнём идти по увеличению t. Для каждого участника i будем поддерживать вероятность q_i того, что он ещё не сдал задачу j, исходно равную 1. Тогда, чтобы обработать очередное t, нужно сделать следующее:

- для каждого участника i добавить к его ответу $p(i, j, t) \cdot \prod_{i_0 \neq i} q_{i_0}$;
- для каждого участника i уменьшить q_i на $p(i, j, t)$.

Чтобы быстро найти произведение q_{i_0} по всем $i \neq i_0$, можно сначала найти произведение всех значений q, а потом делить его на q_i для каждого i. Тогда время работы второй части решения составит $O(nmk)$.

Итоговая сложность решения — $O(nm^2k)$.