

Задача 1. Волшебная традиция

Для сохранения точности при расчетах нужно было использовать целочисленные типы данных, переведя всю сумму из рублей и копеек в только копейки. Это легко можно сделать по формуле $100 \cdot X + YU$. Также, для соответствия ограничениям на количество месяцев и месячным зарплатам, нужно было использовать достаточно большой тип данных, например «long long» в C++.

Для вычисления среднего значения нужно сложить все месячные зарплаты и поделить на количество месяцев. Чтобы учесть округление в сторону работника, нужно прибавить к частному единицу в случае, если деление было выполнено с остатком. Теперь достаточно вывести среднюю зарплату в нужном формате, выразив ответ через рубли — $\lfloor \frac{ans}{100} \rfloor$ и копейки — $ans \% 100$.

Задача 2. Разнообразие десертов

Для решения **подзадачи 2** надо воспользоваться динамическим программированием. Состоянием будет $dp[i][c_1][c_2][c_3]$ — количество способов i людям выбрать десерты так, чтобы десертов первого вида осталось c_1 штук, второго — c_2 , а третьего — c_3 .

Начальное состояние: $dp[0][a_1][a_2][a_3] = 1$ (до прихода людей все десерты на месте).

Переход: Перебираем все 2^N вариантов того, какой набор десертов купил i -й человек (для $N = 3$ это 8 вариантов). Если i -й покупатель взял x_1, x_2, x_3 десертов каждого типа соответственно, то $i-1$ человек до него оставили от изначальных десертов $c_1 + x_1, c_2 + x_2, c_3 + x_3$ десертов соответствующего вида. Если $c_j + x_j \leq a_j$, для $j = 0, 1, 2$, то прибавляем значение $dp[i-1][c_1 + x_1][c_2 + x_2][c_3 + x_3]$ к $dp[i][c_1][c_2][c_3]$.

Для простоты реализации при $N < 3$ можно добавить к существующим десертам еще $3 - N$ видов, количество которых будет по 0 штук.

Решение будет работать за $O(M \cdot A^3)$, где A — это максимальное количество десертов одного вида.

Для решения **подзадачи 3** надо понять, что решение можно делать независимо по каждому виду десертов, а после перемножить результаты для каждого вида. Для этого также воспользуемся динамическим программированием. Состоянием будет $dp[i][j]$ — количество комбинаций при очереди из i покупателей и количестве десертов, равном j .

Начальное состояние: $dp[0][j] = 1$, для $0 \leq j$ (если в кондитерскую никто не придет, то количество десертов не изменится).

Переход: i -й человек либо возьмет десерт, либо нет. Тогда если $j > 0$, то $dp[i][j] = dp[i-1][j-1] + dp[i-1][j]$, иначе $j = 0$ и $dp[i][0] = dp[i-1][0]$.

Решение будет работать за $O(M \cdot A)$, где A — это максимальное количество десертов одного вида.

Для решения **подзадачи 4** надо также, как и в подзадаче 3, решать по каждому виду десертов независимо. То есть для каждого вида с запасом a нужно посчитать количество способов распределить не более a одинаковых десертов среди M различных покупателей.

Количество способов распределить в точности x десертов среди M покупателей является известной комбинаторной величиной — количеством сочетаний из M по x и равно $C_M^x = \frac{M!}{x!(M-x)!}$. Тогда количество способов распределить не более a десертов равно сумме C_M^x , при $0 \leq x \leq \min(a, M)$.

Тогда для ответа на задачу надо посчитать все значения C_M^x , при $0 \leq x \leq M$, а после за $O(M)$ посчитать префикс-сумму этих значений, которая является ответом на задачу про распределение не более a десертов.

Для того, чтобы быстро посчитать C_M^x по модулю $10^9 + 7$, надо предпосчитать значения факториалов $x!$, при $0 \leq x \leq M$, а также найти обратное по модулю $10^9 + 7$ к значению остатка $x!$

(это можно сделать при помощи бинарного возведения в степень $10^9 + 5$ по модулю $10^9 + 7$). Тогда $C_M^x \bmod (10^9 + 7) = (fact[M] \cdot inv[M - x] \cdot inv[x]) \bmod (10^9 + 7)$, где $fact[i]$ — значение $i!$ по модулю $10^9 + 7$, $inv[i]$ — обратное к $fact[i]$ по модулю $10^9 + 7$.

Решение будет работать за $O(N + M \cdot \log(mod))$, где $mod = 10^9 + 7$.

Задача 3. Трибки

Для подзадач 2 и 4, где $\|S\| = 3$, $C = 1$, понятно, что для строки S ответ будет 1, если среди трибок есть строка S , и 0, если среди трибок нет строки S .

Для подзадачи 2 для каждого слова S в списке Бобы можно перебрать все N трибок и проверить, совпадает ли какая-то с S . Это будет работать за $O(N \cdot Q)$, что пройдет эту подзадачу.

Однако такое решение не пройдет по времени для подзадачи 4. Поэтому для решения этой подзадачи сохраним все трибки в множестве (для языка C++ это будет `std::set`, для Python — `set`), и для каждой строки S можем быстро проверять ее наличие среди всех трибок. Это будет работать за $O((N + Q) \cdot \log(N))$.

Для подзадач 3 и 5, где $\|S\| = 3$, меняется только то, что если строка S есть среди трибок, то вместо единицы ответом будет соответствующее значение C . Для подзадачи 3 можно делать аналогично подзадаче 2, и это будет работать за $O(N \cdot Q)$.

Для подзадачи 5 по сравнению с подзадачей 4 меняется только то, что все трибки будем хранить не в множестве, а в словаре, где ключ — трибка, а значение — это количество трибок данного вида (для языка C++ это будет `std::map`, для Python — `dict`). Это будет работать за $O((N + Q) \cdot \log(N))$.

Для остальных подзадач длина строки S может быть больше 3. Поэтому, для каждой строки S найдем все необходимые для него трибки (подстроки длины 3) с учётом количества их вхождения. Так, для строки `abaaba` необходимы 2 трибки `aba`, одна трибка `baa` и одна трибка `aab`. Рассмотрим какую-нибудь трибку, необходимую для построения строки S . Пусть для построения слова необходимо K таких трибок, а всего есть C таких трибок. Тогда невозможно собрать более чем $\lfloor \frac{C}{K} \rfloor$ слов, так как в ином случае трибок для составления слов просто не хватит. Значит, ответом будет минимальное из значений $\lfloor \frac{C}{K} \rfloor$ по всем трибкам, которые необходимы для составления строки S .

Для подзадач 6 и 7 можно для каждой трибки, необходимой для S , искать её количество, как в подзадачах 2 и 4, и это будет работать за $O(N \cdot \sum(\|S\|))$.

Для подзадачи 8 будем для каждой трибки, необходимой для S , искать её количество, как в подзадаче 5, заранее сохранив все трибки в множестве. Такое решение будет работать за $O((N + \sum(\|S\|)) \cdot \log(N))$.

Задача 4. Иерархия

Заметим, что иерархия образует дерево, в которое добавили дополнительные ребра, и задача сводится к вариации поиска наименьшего общего предка двух вершин.

Для первых двух подзадач у каждой вершины не более одного потомка, поэтому получившийся граф на самом деле — линия, причём размер графа не превосходит 61 вершины $10^{18} \leq 2^{60}$, из-за чего найти нужный путь можно любым удобным способом.

В следующих подзадачах нужно научиться находить наименьшего общего предка двух (НОП) вершин в графе. Поскольку граф может быть достаточно большим, явно считать для всего графа НОП не получится. Заметим, что номер каждого сотрудника можно представить в k -ичной системе счисления и НОП двух сотрудников — сотрудник с номером, равным наибольшему общему префиксу. При этом заместитель руководителя может сократить дистанцию на 1 в том случае, если один из сотрудников либо является заместителем общего

руководителя либо является одним из (необязательно непосредственных) подчинённых заместителя.

Задача 5. Иннокентий, люк и треугольники

Для того, чтобы треугольник полностью поместился в окружность, достаточно, чтобы большая сторона не превосходила диаметр окружности. Это верно, поскольку геометрическое место точек вершин треугольников, находящихся напротив большей стороны — круг диаметром с большую сторону, при этом сама окружность — геометрическое место точек вершин прямоугольных треугольников.

Для решения первых двух подзадач явно переберем все 3 стороны треугольника.

Для решения третьей подзадачи заметим, что большая сторона c зависит от двух меньших сторон a и b как $\lceil \sqrt{a^2 + b^2} \rceil \leq c < a + b$, а, значит, количество можно посчитать, как сумму

$$\text{по } a \text{ и } b : \sum_{a=1}^{2 \cdot R} \sum_{b=a}^{2 \cdot R} \min(2 \cdot R + 1, a + b) - \lceil \sqrt{a^2 + b^2} \rceil.$$

Для решения последней подгруппы заметим, что сторона a не превосходит $\sqrt{(2)R}$, а b не превосходит $\lfloor \sqrt{4 \cdot R^2 - a^2} \rfloor$. Также значения $\lceil \sqrt{a^2 + b^2} \rceil$ можно пересчитывать, переходя от b к $b + 1$, что быстрее извлечения корня.