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 $100 \cdot X + YY$. 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 $c_j + x_j \leqslant a_j$, 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 $0 \leqslant j$ (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 $C_M^x = \frac{M!}{x! \cdot (M-x)!}$. Тогда количество способов распределить не более a десертов равно сумме C_M^x, при $0 \leqslant x \leqslant min(a, M)$.

Тогда для ответа на задачу надо посчитать все значения C_M^x, при $0 \leqslant x \leqslant M$, 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 остатков $x!$, при $0 \leqslant x \leqslant M$, 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 $C_M^x \mod (10^9+7) = (fact[M] \cdot inv[M-x] \cdot inv[x]) \mod (10^9+7)$, 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 $10^{18} \leqslant 2^{60}$, 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.

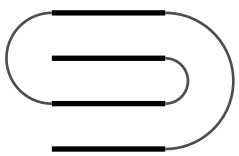
Для решения третьей подзадачи заметим, что большая сторона c зависит от двух меньших сторон a и b как $\lceil\sqrt{a^2+b^2}\rceil \leqslant c < a+b$, а, значит, количество можно посчитать, как сумму по a и b : $\sum_{a=1}^{2\cdot R}\sum_{b=a}^{2\cdot R} \min(2\cdot R+1, a+b) - \lceil\sqrt{a^2+b^2}\rceil$.

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.

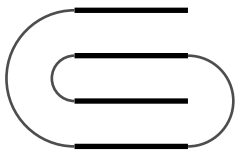
Задача 6. Раз-кладушка, два-кладушка, три-кладушка...

Будем рассматривать каждую укладку, как перестановку чисел от 1 до N в порядке следования экранов ($p[i]$ — позиция i-го экрана, нумеровать позиции будем сверху, начиная с единицы).

Например, вот такую укладку можно рассматривать, как перестановки [2, 3, 1, 4] или [4, 1, 3, 2], в зависимости от того, с какого края укладки начинать:



Если мы повернем эту укладку на 180 градусов, то получим такую проекцию:

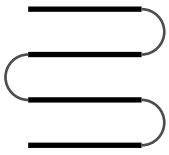


Она задается перестановками [1, 4, 2, 3] и [3, 2, 4, 1].

То есть каждую укладку, которая не имеет вертикальной симметрии, мы можем задать четырьмя перестановками.

Если укладка имеет вертикальную симметрию ($p[i] = N - p[N-i-1]+1$, $i = 0 \ldots N-1$), мы можем задать ее только двумя перестановками.

Например, вот такую укладку мы можем задать только, как [1, 2, 3, 4] и [4, 3, 2, 1]:



Теперь нам нужно отсечь перестановки, которые имеют пересечения линий сгиба. Для этого нужно рассмотреть пары переходов с $p[i]$ на $p[i+1]$ отдельно для четных и нечетных i, так как линии сгиба поочередно находятся то справа, то слева.

Итак, рассмотрим переходы с $p[i]$ на $p[i+1]$ и с $p[j]$ на $p[j+1]$ (у i и j совпадает четность). Так как линии сгиба можно рассматривать без учета их направления, то без ограничения общности будем считать, что $p[i] < p[i+1]$, $p[j] < p[j+1]$, $p[i] < p[j]$. Если в таком случае окажется, что $p[j] < p[i+1]$, то это будет означать, что линии сгиба пересекаются и такую перестановку необходимо отсечь.

Пусть P — количество перестановок без пересечений линий сгиба, из них S — количество перестановок с вертикальной симметрией. Тогда ответом будет $(P-S)/4 + S/2$.

Если перебирать все перестановки и каждую проверять на пересечения линий сгиба за $O(N^2)$, то итоговая асимптотика решения будет $O(N! \cdot N^2)$.

Чтобы успеть посчитать ответ для $N = 14$ в течение тура, необходимо было оптимизировать перебор. Будем строить перестановку рекурсивно, сразу отсекая те ветки, где мы встретили пересекающиеся линии сгиба. Оценить асимптотику решения после такой оптимизации довольно сложно, скажем только, что авторское решение считает ответ для $N = 14$ за 7 секунд. Предпосчитав ответы для всех N от 2 до 14, можно было их сохранить и выводить за константное время.

Задача 7. Эфемерный сад

Для решения подзадачи 2 переберем все $N!$ перестановок цветков, в каком порядке хотим их сорвать. Для каждой перестановки считаем идем по цветкам в порядке в перестановке, для каждого вычисляем минимальное время, в которое можем его сорвать. Если оказывается, что какой-то цветок сорвать не успеем, то дальше можем не идти. Максимальное количество сорванных цветков по всем перестановкам и будет ответом.

Для решения подзадачи 3 будем решать задачу динамикой по подмаскам: $dp[mask][last]$ — минимальный момент времени, к которому можно сорвать все цветки из маски $mask$ так, чтобы последний сорванный цветок был $last$. Ответом будет максимальное такое k, что существует маска $mask$, состоящая из k единиц и последний цветок $last$, что $dp[mask][last]$ существует.

Для решения подзадачи 4 заметим, что максимальный момент времени, когда можем взять цветок, равен $3 + 5 - 1 = 7$. Это означает что после момента 7 ходы путешественника никак не повлияют на ответ. Заметим, что в каждый момент времени у него есть 5 вариантов ходов. Можем считать, что путешественник начинает в момент времени 1, и он начинает в точке, в которой есть или будет цветок (так как даже если $d_i > 1$, то он может вначале стоять на месте). Поэтому просто переберем всевозможные начала путешественника, и всевозможные варианты сделать 6 ходов с момента времени 1 до момента времени 6 включительно. Количество путей, которые надо перебрать, не превышает $50 * 5^6 < 10^6$

Для решения подзадачи 5 заметим, что цветки можем срывать только в порядке возрастания d_i. Поэтому упорядочим цветки в порядке возрастания d_i и будем считать динамику $dp[i]$ — максимальное количество цветков, которые можно сорвать, если последний сорванный цветок имеет номер i. Ответ — максимальное из всех $dp[i]$.

Для решения 6-ой подзадачи каждый цветок с $t_i = 2$ разделим на 2 цветка с временами появления d_i и $d_i + 1$. Упорядочим получившиеся цветки и также считаем $dp[i]$. Отличие состоит в том, что при вычислении $dp[i]$ не учитываем $dp[j]$, если цветкам i, j после разделения соответствовал один и тот же цветок до разделения. Это нужно, чтобы один и тот же цветок не учесть дважды.

Для решения подзадачи 7 уже не можем просто использовать предыдущую идею, так как

динамика может посчитать срыв одного цветка дважды, даже если между ними был срыв другого цветка (например, сорвали цветок 1 с $t_1 = 3$ в момент времени d_1, на следующий момент времени перешли в соседнюю точку и сорвали цветок 2, в момент времени $d_1 + 1$, а далее вернулись к цветку 1 в момент времени $d_1 + 2$, что не превосходит $d_1 + t_1 - 1$). Поэтому решать задачу будем другой динамикой: $dp[cnt][last2][last1]$ — минимальное время, к которому можем сорвать cnt цветков, предпоследний сорванный цветок был $last2$, последний сорванный — $last1$ (при $cnt = 1$ значение $last2$ может быть любым) Пересчитываем в порядке возрастания cnt, через $dp[cnt - 1][last3][last2]$, по всем $last3$ таким, что $last3$ не равно $last1$ и не равно $last2$. Ответом будет максимальное такое cnt, что существуют $last2$ и $last1$, для которых $dp[cnt][last2][last1]$ существует.

Для решения подзадачи 8 можем вместо двух последних сорванных цветков хранить 3, то есть хранить такую динамику: $dp[cnt][last3][last2][last1]$. Однако можно заметить, что динамика для подзадачи 7 также должна проходить и подзадачу 8. Действительно, единственная казалось бы возможная проблема — то, что один и тот же цветок мы можем учесть дважды. То есть, при пересчете $dp[cnt][last2][last1]$ через $dp[cnt - 1][last3][last2]$ получили, что цветок $last1$ был посещен дважды. Так как $t_i \leqslant 4$, то возможен лишь такой путь: $last1$ — $last3$ — $last2$ — $last1$, причем каждый переход от цветка к цветку занимал 1 единицу времени. Тогда раскрасим точки поля в шахматном порядке, и каждый переход в соседнюю точку менял цвет. То есть цветки $last1$ и $last3$ стоят на точках разных цветов, также точки $last3$ и $last2$ и точки $last2$ и $last$. Но так как цветов всего 2 — данная ситуация невозможна. Значит, при такой динамике мы не учтём один и тот же цветок дважды.

Для решения подзадачи 9 хранить явно последние 4 сорванных цветка в динамике $dp[cnt][last4][last3][last2][last1]$ невозможно из-за ограничений по времени и памяти. Поэтому возможны два подхода:

1. Неявно хранить такую динамику и рекурсивно пересчитывать.

2. Использовать другую идею для динамики.

Для второго подхода заметим, что для того, чтобы избежать случая повторного срыва цветка, вместо хранения информации о нескольких последних сорванных цветках можем лишь хранить информацию о последнем сорванном цветке и о последних ходах. Ходы хранить выгоднее, так как вариантов ходов всего 5, когда количество цветков может достигать 50. Так как $t_i \leqslant 5$, то достаточно лишь хранить информацию о последних 4 ходах. Поэтому можем использовать динамику: $dp[cnt][last][mask_{moves}]$ — минимальное время к которому можем сорвать cnt цветков, последний сорванный цветок имеет номер $last$, а $mask_{moves}$ хранит последние 4 хода и может принимать одно из 5^4 значений. Ответом будет такое максимальное такое cnt, что существует $last$ и $mask_{moves}$, что существует $dp[cnt][last][mask_{moves}]$.

Задача 8. Совет Четырёх Башен

Для решения подзадачи 2 можно для каждого мага i определить, является ли он бесперспективным или нет. Для этого достаточно перебрать всех остальных магов $j \neq i$, и проверить, занял ли маг j лучшее место в каждом испытании, чем маг i.

Решение будет работать за $O(N^2)$, что будет проходить эту подзадачу.

Для решения подзадачи 3 заметим, что при заданных ограничениях можно считать, что есть всего 2 стихии (соответствующие c_i и d_i) Для этого переупорядочим всех магов так, чтобы значения c_i шли по возрастанию. i-ый маг будет бесперспективным, если есть такой маг j, что $j < i$ и $d_j < d_i$. Такой маг будет существовать, если $d_i > \min(d_1, d_2, \ldots, d_{i-1})$. Остается

пройтись по возрастанию i, на каждом шаге проверять предыдущее условие и обновлять минимум.

Решение будет работать за $O(N \cdot \log(N))$ или $O(N)$ в зависимости от реализации.

Для решения подзадачи 4 заметим, что при заданных ограничениях можно считать, что есть всего 3 стихии (соответствующие b_i, c_i и d_i). Для этого переупорядочим всех магов так, чтобы уже значения b_i шли по возрастанию. i-ый маг будет бесперспективным, если есть такой маг j, что $j < i$, $c_j < c_i$ и $d_j < d_i$. Получается, что понадобится структура данных, которая умеет быстро выполнять эти два типа операций:

1. Добавлять пару (c_i, d_i) в структуру, $(1 \leqslant c_i, d_i \leqslant N)$
2. Для пары (x, y) определить, существует ли хоть одна уже добавленная пара (c_i, d_i), что $x < c_i$ и $y < d_i$, $(1 \leqslant x, y \leqslant N)$

Назовем для второго типа операции пару (x, y) *плохой*, если будет существовать подходящая пара и *хорошей* в противном случае. Заметим, что для любого x и y верно, что если пара (x, y) — плохая, то и пара $(x, y+1)$ — плохая. Также, если для пары $(x, y+1)$ — хорошая, то и пара (x, y) будет хорошей. Это означает, что для любого x существует такое число S_x, что для любого $y \leqslant S_x$ пара (x, y) является хорошей, а для любого $y > S_x$ пара (x, y) является плохой. Вначале, когда в структуре нет ни одной пары, любая пара (x, y) является хорошей, значит, $S_x = N$ для любого x. Рассмотрим, что произойдет с S при добавлении пары (c_i, d_i).

Заметим, что для $x \leqslant c_i$ добавленная пара никак не повлияет на то, будет ли пара хорошей или нет, и S_i останется прежним. Для $x > c_i$ пара (c_i, d_i) будет подходящей, если $d_i < y$. Поэтому S_x меняется на $min(S_x, d_i)$. Для того, чтобы проверить, является ли пара (x, y) хорошей, достаточно проверить, выполняется ли, что $y \leqslant S_x$. Для быстрого выполнения этих операций подойдет дерево отрезков либо дерево Фенвика.

Решение будет работать за $O(N \cdot \log(N))$.

Для решения подзадачи 5 для каждого мага i вычислим три значения:

1. num_i — номер испытания (от 1 до 4), в котором маг i занял другое место от других трех испытаний. Если маг занял одинаковое место во всех четырех испытаниях, то без ограничения общности можем считать, что это первое испытание.
2. V_i — место, которое он занял в этом испытании.
3. T_i — место, которое он занял в каждом из трех других испытаниях.

Рассмотрим магов i и j, и выясним, когда маг j показал лучшее место, чем маг i в каждом из четырех испытаний. Если $num_i = num_j$, то необходимо и достаточно, чтобы выполнялось $T_j < T_i$ и $V_j < V_i$ Если же $num_i \neq num_j$, то необходимо и достаточно, чтобы выполнялось $V_j < T_i$, $T_j < V_i$ и $T_j < T_i$. Перепишем в этом случае условия на T_j, V_j: $T_j < min(V_i, T_i)$, $V_j < T_i$. Воспользуемся структурой данных из подзадачи 4. Будем хранить четыре такие структуры, k-ая будет хранить пары (T_i, V_i) для тех магов i, что $num_i = k$. Чтобы проверить, является ли рассматриваемый маг i бесперспективным, достаточно для структуры под номером num_i выяснить, является ли пара (T_i, V_i) хорошей, а для остальных трех структур выяснить, является ли пара $(min(V_i, T_i), T_i)$ хорошей. Если хоть раз из четырех пара окажется плохой, то маг является бесперспективным.

Решение будет работать за $O(N \cdot \log(N))$.

Для полного решения вначале переупорядочим всех магов так, чтобы уже значения a_i шли по возрастанию. i-ый маг будет бесперспективным, если есть такой маг j, что $j < i$, $b_j < b_i$,

$c_j < c_i$ и $d_j < d_i$. Задачу будем решать методом «разделяй и властвуй». Для каждого мага i будем поддерживать, нашли ли такого мага j, из-за которого маг i стал бесперспективным. Пусть мы хотим решить задачу для отрезка с мага i по маг j. Если $i = j$, то на отрезке всего один маг, и ничего не надо делать. Если $i < j$, то разобьем на два отрезка: от мага i до мага k и от мага $k+1$ до мага j таким образом, что количества магов в этих двух отрезков отличаются не более чем на 1. Для каждого из двух отрезков задачу решим рекурсивно. Заметим, что на перспективность магов из левого отрезка не могут повлиять маги из правого отрезка, так как в первом испытании у магов из левого отрезка лучшие места. Далее для каждого мага из правого отрезка найдем, есть ли маг из левого отрезка, который делает того мага бесперспективным. Заведем структуру из подзадачи 4, в которой будем хранить пары (c_i, d_i). Чтобы правильно учитывать значения b_i, будем идти двумя указателями по магам из левого и правого отрезков. Для метода двух указателей необходимо, чтобы левый и правый отрезки были упорядочены по возрастанию b_i, что достигается за счет того, что после вычислений мы мерджим левый и правый отрезки (как при сортировке слиянием). Несмотря на то, что мы меняем местами магов, условие на a_i будет также корректным, так как маги внутри и левого, и правого отрезка менялись местами исключительно внутри того отрезка, в котором они были.

Решение будет работать за $O(N \cdot \log^2(N))$.