Задача A. Проблемы с костюмом

Автор задачи: Михаил Иванов, разработчик: Даниил Орешников

Заметим, что количество различных наборов конечностей и количество различных наборов голов, которые можно заставить, не зависят друг от друга. Поскольку матожидание независимых случайных величин равно произведению их матожиданий, мы можем независимо посчитать матожидания количеств различных наборов конечностей и различных наборов голов. Ответом тогда будет их произведение.

Найдем матожидание количества различных наборов конечностей, для голов будет аналогично. Разобьем все n^a возможных наборов конечностей на группы с общим *профилем*. Профиль — последовательность, описывающая совпадения и различия между элементами набора. На первом месте в профиле всегда стоит число 1, а на каждом следующем — либо уже встречавшееся раньше число, означающее, что тип соответствующей конечности совпадает с уже встреченной ранее, либо число, на 1 большее максимального из уже встречавшихся, означающее, что эта конечность какого-то нового типа.

Например, профиль $\langle 1, 2, 3, 2, 4\rangle$ описывает последовательности $[1, 2, 3, 2, 4]$ или $[7, 4, 11, 4, 1]$, но не $[1, 2, 3, 2, 3]$, так как последний элемент должен отличаться от всех предыдущих. А, например, $\langle 1, 2, 1, 2, 4\rangle$ вообще не является описанием профиля, потому что новый тип должен обозначаться числом, на 1 большим предыдущего максимума.

Сгенерировать все профили длины 5 под a типов конечностей можно следующим образом:

```
gen(profile):
    if size(profile) == 5:
        запомнить profile
    else:
        for i = 1 ... max(profile) + 1:
            gen(profile + [i])
```

Заметим, что наборы конечностей с различными профилями сами по себе отличаются, а наборы с одинаковым профилем равновероятны, так как каждый тип конечности встречается равновероятно. Из второго утверждения следует, что матожидание количества различных наборов с фиксированным профилем `profile` равно вероятности получить конкретный набор p с таким профилем, умноженной на количество различных наборов с таким профилем, а тогда

$$\texttt{Answer} = \sum_{\texttt{profile}} \text{count}(\texttt{profile}) \cdot \text{prob}(p)$$

Посчитаем количество различных наборов с заданным профилем. Вместо каждого уникального числа можно подставить любой уникальный тип, поэтому

$$\text{count}(\texttt{profile}) = n \cdot (n-1) \cdot \ldots \cdot (n - \max(\texttt{profile}) + 1)$$

В качестве конкретного представителя профиля возьмем набор, в котором в профиле вместо числа i подставили тип i, и посчитаем вероятность иметь возможность получить такой набор. Например, профиль $\langle 1, 2, 3, 1, 2\rangle$ представим набором $[1, 2, 3, 1, 2]$. Его можно получить тогда и только тогда, когда среди a доставленных конечностей есть хотя бы две типа 1, хотя бы две типа 2 и хотя бы одна типа 3. Вероятность этого равна $1 - \text{prob}\left(cnt(1) \leqslant 2 \text{ or } cnt(2) \leqslant 2 \text{ or } cnt(3) \leqslant 1\right)$.

Такую вероятность можно посчитать по *формуле включений-исключений* перебором 2^5 пересечений соответствующих событий вида «ровно столько объектов выбранных типов». Вероятность каждого такого события может быть посчитана по *формуле Бернулли*, например, для события «ровно две конечности типа 1 и одна конечность типа 2» вероятность будет равна

$$C_a^2 \cdot \left(\frac{1}{n}\right)^2 \cdot C_{a-2}^1 \cdot \left(\frac{1}{n}\right)^1 \cdot \left(\frac{n-2}{n}\right)^{a-3}$$

Зная для каждого профиля вероятность возможности получить конкретный описываемый им набор, и количество соответствующих профилю наборов, можно просто посчитать ответ по первой

формуле наверху. Аналогичные вычисления делаем для наборов голов. Умножение по модулю можно выполнять, используя `long long`, деление по модулю — это умножение на обратный, который можно найти, используя *малую теорему Ферма* $a^{-1} \equiv a^{p-2} \mod p$, и *быстрое возведение в степень* за $\log p$:

```
power(a, d):
    if d == 0:
        return 1
    if d % 2 == 1:
        return power(a, d - 1) * a % p
    sqrt = power(a, d / 2)
    return sqrt * sqrt % p
```

Задача B. Устрашающий палиндром

Автор задачи: Григорий Хлытин, разработчик: Мария Жогова

Заметим, что если все данные нам строки, расположенные в некотором порядке, дают палиндром, то первая строка является разворотом последней, вторая — разворотом предпоследней, и так далее. Например, если в начале стоит «`abc`», то в конце обязана стоять «`cba`», являющаяся ее разворотом.

Посчитаем от каждой строки прямой и обратный *полиномиальный хэш*. Если одна строка является разворотом другой, то ее прямой хэш будет совпадать с обратным хэшом второй, и наоборот. Создадим для каждой строки s_i два объекта: $(\text{fhash}(s_i), i, 0)$ и $(\text{rhash}(s_i), i, 1)$, где первый элемент — прямой и обратный хэш соответственно, затем номер строки, и затем флаг, указывающий на то, прямой или обратный хэш хранится в первом элементе.

Все такие объекты разобьем на группы с одинаковым первым элементом. Это можно сделать с помощью `unordered_map`. Пусть $g(h)$ — группа (множество) всех объектов с хэшом, равным h. Заметим, что в каждой такой группе (кроме, возможно, одной, если n нечетно), все объекты должны разбиваться на пары из прямого и обратного хэша разных строк, чтобы эти строки можно было расположить симметрично относительно середины палиндрома. Чтобы это проверить, достаточно пройтись по каждой группе, и жадно сопоставить каждому объекту первый встреченный объект с противоположным направлением и другим номером строки.

Если в какой-то группе после такого сопоставления остались объекты без пар, то

1. Либо в группе было непоровну прямых и обратных хэшей, в случае чего ответа не существует;
2. Либо прямой и обратный хэш какой-то из строк совпали, тогда она сама является палиндромом. В случае четного n это означает, что ей в пару можно было сопоставить только такую же строку, а раз соответствующий объект остался без пары, то такой нет. В случае нечетного n такая строка обязана остаться ровно одна, чтобы встать в центр палиндрома.

Таким образом, достаточно жадно сопоставить объекты внутри каждой группы, и проверить, что осталось ровно $n \bmod 2$ строк без пары, которые будут расположены в центре ответа. После чего можно сначала вывести номера первых строк из каждой пары, а затем в обратном порядке номера вторых строк из пар.

Задача C. Патруль экзорцистов

Автор задачи: Михаил Иванов, разработчик: Арсений Кириллов

Так как по условию даны n площадей и $n-1$ улиц между ними, то структура этих площадей представляет собой дерево из n вершин.

Посмотрим на вершину запроса v и подвесим дерево за неё. Теперь вершина v — корень дерева, а значит все оставшиеся вершины — ее потомки. Посмотрим на какого-нибудь сына u вершины запроса. Если в поддереве вершины u есть вершина, которая находится на расстоянии большем d от вершины v, то в этом поддереве точно надо удалить хотя бы одно ребро. Если все поддерево останется нетронутым, то эта вершина все еще будет достижима.

Поэтому, если в поддереве вершины u есть «далекая» от v вершина, удалим ребро (v, u), тогда ни одна вершина из этого поддерева не будет достижима из вершины v, в частности, не будут достижимы вершины на расстоянии больше d. Если же в этом поддереве нет такой вершины, то и удалять рёбра из него не нужно, потому что поддеревья разных детей независимы между собой.

Ограничение по времени, правда, не позволяет для каждого запроса честно переподвешивать дерево за вершину из запроса, после чего проверять описанное выше свойство для всех ее детей. Подвесим дерево за вершину 1, после чего посчитаем для каждой вершины $\texttt{dp1}[v]$ — максимальную длина пути из v в ее потомка, и $\texttt{dp2}[v]$ — максимальную длину пути из v, первое ребро которого ведет из v наверх по дереву. Будем считать обе динамики с помощью *обхода в глубину*.

Первым обходом в глубину посчитаем $\texttt{dp1}$ — для этого после того, как из вершины v посещены все ее дети, отсортируем их по возрастанию $\texttt{dp1}$, после чего запишем

$$\texttt{dp1}[v] = \max_{u \in \text{children}(v)} \texttt{dp1}[u] + 1$$

.

Вторым обходом посчитаем $\texttt{dp2}$. Максимальный из путей «вверх» по ребру $v \to w$ либо идет «вверх» из w, либо идет из w в ее поддерево, но **не по ребру** $w \to v$. Первое значение — это просто $\texttt{dp2}[w]$, а второе — максимальное из $\texttt{dp1}$ детей w, не равных v, плюс 1 (что можно найти за $\mathcal{O}(1)$, так как дети каждой вершины отсортированы по возрастанию $\texttt{dp1}$):

$$\texttt{dp2}[v] = \max\left(\texttt{dp2}[w], \max_{\substack{u \in \text{children}(w) \\ u \neq v}} \texttt{dp1}[u] + 1\right) + 1$$

Посчитав две такие динамики, можно за $\mathcal{O}(\log n)$ определять, по скольким ребрам из вершины достижимы вершины на расстоянии больше d: для пути «вверх» достаточно сравнить $\texttt{dp2}[v]$ и d, а для путей «вниз» — сделать бинпоиск по детям вершины, так как они отсортированы по $\texttt{dp1}$.

Задача D. Гигаскелеты

Автор задачи и разработчик: Даниил Голов

Несложно доказать, что множество может быть *НОК-широким* тогда и только тогда, когда в нем есть число, делящее все остальные. Действительно, пусть это не так. Тогда рассмотрим два максимальных различных числа a_i и a_j из множества. Если ни одно из них не делится на другое, то $d = \text{НОК}(a_i, a_j)$ больше каждого из них, а значит в множестве нет a_k, не меньшего d (так как были выбраны максимальные элементы).

Теперь задача свелась к тому, чтобы разбить числа на множества, в каждом из которых есть элемент, делящий любой другой, чтобы сумма НОК по этим множествам была минимальна. Заметим, что в таком случае если есть a_i, делящее какое-либо другое число, оно должно быть в группе с одним из делящихся на него чисел. Иначе их группы можно объединить, уменьшив суммарный НОК. Аналогично, если есть число, не делящее никакое другое, оно гарантированно будет максимумом в своей группе, и НОК в ней будет равен ему.

Поэтому, требуется найти для каждого числа любое, делящееся на него, и добавить его в соответствующую группу. С помощью *решета Эратосфена* можно для каждого числа найти минимальное простое в его составе, а, следовательно, и его полноценное разложение на простые. После чего можно построить граф на числах от 1 до $A = \max(a)$, в котором будет проведено ребро $x \to y$, если y получается делением x на некоторое простое. Так как простых в составе числа не больше логарифма, размер такого графа составит не более $A \log A$ ребер, а значит построение графа и *обход в глубину* укладываются в поставленные ограничения по времени.

С помощью обхода в глубину для каждого числа из ввода можно определить, есть ли во вводе большее его число, делящееся на него (критерий — если число было посещено `dfs`, запущенным из другого числа), и таким образом сгруппировать числа в НОК-широкие группы.

Задача E. Нужно меньше дорог!

Автор задачи и разработчик: Владислав Власов

В задаче дан граф (не обязательно связный) и требуется добиться того, чтобы между любой парой вершин существовало не более одного пути. Давайте решим задачу отдельно для каждой компоненты связности.

Так как нам нужно чтобы как можно больше пар домов остались достижимы друг из друга, нельзя разделять компоненту связности на несколько новых. А из единственности пути между любой парой домов следует, что компоненту необходимо привести к дереву, то есть построить ее остов и удалить все оставшиеся ребра.

Если бы не любимые дороги жителей, можно было бы обойти компоненту поиском в глубину и выкинуть ребра, по которым не были совершены переходы. Так как они все же есть, и каждая такая дорога обязана войти в ответ, первым шагом запустим *обход в глубину* только по неудалямым ребрам и выделим «неудаляемые» компоненты связности. Если какие-то из таких компонент содержат цикл, то ответ не существует, это можно проверить тем же поиском в глубину.

Если же все неудаляемые компоненты не содержат циклы, сожмем каждую из них в одну вершину, после чего запустим рассмотренное в начале решение для случая, когда неудаляемых ребер нет. Как сжатие, так и обход в глубину, занимают время $\mathcal{O}(n)$.

Также задачу можно решить модификацией *алгоритма Краскала* за $O(m\alpha(n))$, отдав приоритет неудаляемым ребрам (рассмотрев их в любом порядке), а затем в любом порядке рассмотрев все оставшиеся.

Задача F. Ритуал очищения

Автор задачи и разработчик: Даниил Орешников

Для начала заметим, что существам выгодно подходить к чаше в порядке увеличения (невозрастания) их оставшихся сроков проклятий. Действительно, пусть $a_i > a_j$, и j-е существо подошло к чаше сразу после i-го, и после этого оба существа полностью очистились. Если в чаше перед этим было x песчинок, теперь их стало

$$A = (x^2 - a_i)^2 - a_j = x^4 - 2x^2 a_i + a_i^2 - a_j$$

Если бы существа подошли в противоположном порядке, то, аналогично, получилось бы число песчинок $B = x^4 - 2x^2 a_j + a_j^2 - a_i$. Разность двух этих чисел равна $A - B = (x^4 - 2x^2 a_i + a_i^2 - a_j) - (x^4 - 2x^2 a_j + a_j^2 - a_i) =$

$$(a_i - a_j) \cdot (-2x^2 + (a_i + a_j) + 1)$$

Поскольку $a_i > a_j$, первый множитель положителен. А поскольку по нашему предположению в первом случае оба существа полностью очистились, $x^2 \geqslant a_i$, то есть $x^2 > a_j$, а тогда $1 + a_i + a_j - 2x^2 \leqslant 0$. Это означает, что $A - B \leqslant 0$, то есть $A \leqslant B$. А тогда, если бы j-е существо подошло к чаше **перед** i-м, в чаше бы осталось не меньше песчинок, чем если бы они подошли в противоположном порядке.

Таким образом, если x песчинок хватает для очищения всех существ, то их хватает и в том случае, если все существа будут подходить в порядке возрастания a_i. Отсортируем существ по возрастанию a_i. Теперь посчитаем, сколько необходимо песчинок, чтобы все смогли очиститься:

1. чтобы последнее существо могло очиститься, перед его подходом должно быть хотя бы $x_n = \lceil \sqrt{a_n} \rceil$ песчинок;
2. чтобы после предпоследнего существа осталось хотя бы x_n песчинок, перед его подходом их должно быть хотя бы $x_{n-1} = \lceil \sqrt{a_{n-1} + x_n} \rceil$;
3. аналогично, перед предыдущим их должно быть хотя бы $x_{n-2} = \lceil \sqrt{a_{n-2} + x_{n-1}} \rceil$;

   ...

n. перед первым существом должно быть хотя бы $x = \lceil \sqrt{a_1 + x_2} \rceil$ песчинок.

Итого, пройдясь по массиву a в порядке убывания сроков, мы сможем посчитать, сколько песчинок должно быть в чаше изначально, чтобы все существа могли снять свои проклятия. Время работы — $\mathcal{O}(n \log n)$ (из-за того, что массив надо отсортировать).

Задача G. Потрошение вывески

Автор задачи и разработчик: Даниил Орешников

Решим задачу, используя метод *динамического программирования*. Заметим, что есть два принципиально разных случая — когда первое потрошение произошло по строке, и когда оно произошло по столбцу. Если первое потрошение было по строке, то в таблице не может быть выделен ни один целый столбец, и наоборот. Обозначим количество распотрошить таблицу $n \times m$, начиная со строки, за $\texttt{dp1}[n][m]$, и со столбца — за $\texttt{dp2}[n][m]$.

Рассмотрим случай первого потрошения по строке. Заметим, что если несколько строк выделены целиком в результате потрошений по строкам, то не важно, в каком порядке эти потрошения были сделаны, результат будет один и тот же. Рассмотрим самую верхнюю строку, по которой было сделано потрошение, пусть у нее номер i. Тогда в верхней части первое потрошение гарантированно сделано по столбцу (так как мы выбрали самую верхнюю выбранную целиком строку), а в нижней — в любом направлении. Тогда количество способов распотрошить оставшуюся таблицу равно

$$\texttt{dp2}[i-1][m] \cdot (\texttt{dp1}[n-i][m] + \texttt{dp2}[n-i][m])$$

Аналогичную величину можно посчитать и для первого потрошения по столбцу. Получится следующее соотношение:

$$\texttt{dp1}[n][m] = \sum_{1 \leqslant i \leqslant n} \texttt{dp2}[i-1][m] \cdot (\texttt{dp1}[n-i][m] + \texttt{dp2}[n-i][m])$$

$$\texttt{dp2}[n][m] = \sum_{1 \leqslant j \leqslant m} \texttt{dp1}[n][j-1] \cdot (\texttt{dp1}[n][m-j] + \texttt{dp2}[n][m-j])$$

Базу динамики можно вывести следующим образом: мы хотим, чтобы получались значения $\texttt{dp1}[1][j] = 1$ и $\texttt{dp2}[i][1] = 1$. В общем случае для этого надо присвоить $\texttt{dp1}[0][j] = \texttt{dp2}[i][0] = 0$ и $\texttt{dp1}[i][0] = \texttt{dp2}[0][j] = 1$ для любых i и j.

Осталось только пересчитать эту динамику в двух вложенных циклах по i и j, получим время работы $\mathcal{O}(nm(n+m))$.

Задача H. Парное пугание

Автор задачи и разработчик: Мария Жогова

Давайте представим отношение «*стоять в паре*» в виде графа. Ясно, что такое отношение симметрично, значит граф будет неориентированным. Так как каждый ребенок состоит или в одной паре, или в k, значит и степень каждой вершины такого графа или 1, или k. Из первого условия интересности набора пар следует, что такой граф не имеет циклов. А из второго следует, что граф связен, то есть из **любой** вершины графа, есть путь в **любую** другую.

Так как граф неориентированный и не имеет циклов, его можно представить в виде дерева. Причем листьями такого дерева будут дети, которые состоят только в одной паре, а ветвями (не листьями) — дети, входящие в k ($k \geqslant 2$) пар.

В таком случае задача сводится к построению дерева, соответствующего заданным условиям, и вычислению количества листьев в таком графе.

По условию, $n \geqslant 3$, значит среди вершин нашего дерева есть хотя бы один не лист. Давайте среди всех листов выберем любой, назовем его корнем и подвесим за него наше дерево. Заметим, что теперь любая вершина, кроме корня, имеет $k-1$ или ноль ведущих к детям этой вершины ребер (вершинам, находящимся ниже данной). Ясно, что ребер, ведущих к детям не из корня, в точности $n-1-k$ (в дереве всего $n-1$ ребро и k из них идут из корня). Кроме того, количество таких ребер кратно $k-1$ (действительно, сумма чисел кратных $k-1$ или ноль кратна $k-1$). **Следовательно, такое разбиение на пары существует, если $n-k-1$ кратно $k-1$, то есть если $n-2$ кратно $k-1$.**

Допустим, интересующее нас разбиение на пары существует. Тогда количество не-листьев в точности равно $1+\dfrac{n-k-1}{k-1}$ (количество вершин, не являющихся корнями, у которых есть $k-1$ ребенок плюс корень). А значит количество листьев равно $n-1-\dfrac{n-k-1}{k-1} = n-\dfrac{n-2}{k-1}$.

Время работы — $\mathcal{O}(1)$.

Задача I. Груша на Хэллоуин

Автор задачи и разработчик: Мария Жогова

Для начала заметим, что эта задача сводится к задаче на массиве, так как в дереве существует единственный путь между каждой парой вершин. После этого воспользуемся классической идеей — собирать ответ по битам.

Рассмотрим k-й бит ответа. Он равен 1, если количество пар, таких что $(a_i + a_j) \mod 2^{k+1} > 2^k$, нечетное. Научимся проверять, для каких i и j сумма $a_i + a_j$ имеет единицу в k-м бите. Для этого рассмотрим все числа по модулю 2^{k+1}. Заметим, что $(a_i \bmod 2^{k+1}) + (a_j \bmod 2^{k+1})$ лежит между 0 и $2^{k+2} - 2$, то есть содержит 1 в k-м бите, если лежит в одном из двух интервалов: $[2^k, 2^{k+1})$ или $[2^{k+1} + 2^k, 2^{k+2})$.

Сделаем *цифровую сортировку*. На первой итерации отсортируем все a_i по их остатку при делении на 2, то есть по последнему биту. В таком массиве с помощью *двух указателей* за линейное время можно найти количество пар (i, j), удовлетворяющих описанному выше условию, что дает нам значение 0-го бита ответа. Проведем еще одну итерацию цифровой сортировки, перейдя к остаткам по модулю 2^2, снова пройдемся двумя указателями, и так далее, пока не будут рассмотрены все биты.

Время работы — $\mathcal{O}(n \log(\max a_i))$.

Задача J. Идеальное покрытие треугольниками

Автор задачи и разработчик: Константин Бац

Давайте решать задачу для каждого запроса отдельно.

Заметим, что если существует хотя бы одна плитка со стороной 2^x, то любой квадрат со стороной 2^y при $y < x$ можно покрыть треугольниками с катетами 2^y. Действительно, давайте делить большой треугольник на маленькие до тех пор, пока не получим 4^{x-y} треугольников с катетами 2^y. Из всех получившихся треугольников, а их, напоминаем, не меньше четырех, возьмем два и покроем ими площадь. Ясно, что треугольниками большего размера площадь не покрыть. Давайте в качестве x возьмем максимальный a_i, и тогда для всех $b_j < x$ ответом будет b_j.

Что делать, если $b_j \geqslant x$? Давайте жадным образом брать плитки всех размеров от максимального до минимального, вычитая из необходимого количества плиток количество имеющихся. Заведем переменную `need` $= 2$ — количество плиток размера $k = b_j$, необходимое, чтобы покрыть плиткой непокрытую часть квадрата.

Запустим цикл по k с начальным значением $k = b_j$. На каждой итерации цикла будем вычитать из `need` количество треугольников с катетом 2^k. Если после этого `need` $\leqslant 0$, значит плиток размера не меньше k, разделенных на треугольники с катетами 2^k, хватит.

Если это не так, значит площадь придется покрывать треугольниками меньшего размера. Попробуем сделать все то же самое, но с плитками размера $k-1$. Для этого `need` необходимо увеличить в четыре раза, так как треугольников со стороной в два раза меньше требуется в четыре раза больше. При этом плитки большего размера заново учитывать не нужно, ведь при разбиении на плитки меньших размеров, занимаемая ими площадь не изменится. Цикл нужно остановить, когда $k < 0$ (плиток с такими размерами не бывает) или когда `need` $> 2 \cdot 10^5$. Этого достаточно, так как плиток любого размера не бывает больше, чем $2 \cdot 10^5$, все треугольники большего размера мы уже учли, а если использовать плитки меньшего размера, потребуется большее $4 \cdot 2 \cdot 10^5$ плиток.

Итого, для каждого из m запросов мы выполним не больше, чем $\log(2 \cdot 10^5) \approx 10$ итераций подбора максимального размера плитки. Время работы — $\mathcal{O}(n + m)$.