Задача A. Изгороди

Так как концы изгородей находятся на расстоянии n от точки пересечения канав, то изгороди образуют треугольник, вписанный в окружность с центом в этой точке. Так как длинная сторона является диаметром окружности, то искомый угол всягда прямой.

Задача B. Анализ крипторынка

Решение задачи сводится к решению уравнения $a_1^N + a_2^N = a_3^N$ в целых числах. Поле разбора слишком мало, чтобы вместить доказательство отсутствия решений для $N > 2$.

Задача C. Взять след!

Пусть a'_i — последовательность чисел из a_i, упорядоченных по невозрастанию. Для матрицы размера $k \times k$ максимальный след можно получить взяв $a'_1, a'_2, \dots a'_k$. При этом, если $a'_k < 0$, то максимальный след матрицы размера $k-1 \times k-1$ будет больше. А если $a'_{k+1} > 0$, то выгоднее взять матрицу размера $k+1 \times k+1$.

Таким образом, выгодно взять максимальное k, такое что $k \leqslant \sqrt{n}$ и $a'_k \geqslant 0$. Если же $a'_1 < 0$, то $k = 1$ и след будет меньше нуля.

Задача D. Градусы, радианы, грады

Заметим, что однозначно восстановить n и единицу измерения невозможно не только при $n = 0$, но и при $n = 9k$, так как в этом случае число градов будет равно $10k$.

Для решени задачи можно было либо аккуратно разобрать случаи, либо сгенерировать все варианты для $n = 0..359$ и отобрать из них, отличающиеся от числа Алисы менее чем на $5 \cdot 10^{-7}$.

Задача E. Подстроки и подпоследовательности

Заметим, что $S_1 = S_2$ для строк только двух видов: состоящих из повторения одного символа ($\alpha\alpha....\alpha$) и из повторения одного символа, за которым следует повторение другого символа ($\alpha\alpha....\alpha\beta\beta....\beta$).

Для $n \leqslant 10^6$ максимальная длина необычной строки равна 56 и таких строк порядка двух миллионов, то можно было сгенерировать их все, отсортировать и выбрать n-ю из них.

Задача F. Если сложить...

Как известно, корни простых чисел линейно независимы над полем рациональных чисел. То есть никакая сумма таких корней не может дать ни 0, ни любое другое рациональное число. Доказательство этого можно найти в работах `https://www.turgor.ru/lktg/2020/1/1-radicals-ru-sol.pdf` и `https://math.uchicago.edu/ may/VIGRE/VIGRE2007/REUPapers/FINALAPP/Jaffe.pdf`.

Таким образом, в задаче было достаточно извлечь из каждого a_i корень максимально возможной степени. При этом, так как $2^{30} > 10^9$, то при $a_i \geqslant 2$ достаточно рассмотреть корни степени не выше 29. Их можно было либо найти бинарным поиском, либо посчитать с использованием чисел с плавающей точкой.

Задача G. Генерация ключей

Пусть $\overline{N_1...N_q}$ запись числа N в двоичной системе счисления. Будем искать ключи M так же в виде двоичной записи $\overline{M_1...M_q}$.

Рассмотрим ключи, которые первый раз отличаются от N в a-м бите, то есть $N_1 = M_1, N_2 = M_2, \dots N_{a-1} = M_{a-1}$ и $N_a \neq N_b$. Так как $M < N$, то $N_a = 1$ и $M_a = 0$. Таким образом, у M есть $q - a$ «свободный» бит, среди которых должно быть $K - b$ единичных, где b — число единичных бит на общем префиксе N и M. Это можно сделать C_{l-a}^{K-b} способами, где C_n^k — число сочетаний из n по k.

Таким образом, ответом на задачу является сумма C_{l-a}^{K-b} для всех a, для которых $N_a = 1$.

При заданных ограничениях число сочетаний можно было считать либо в длинной арифметике по формуле $C_n^k = \frac{n!}{k!(n-k)!}$, либо треугольником Паскаля по модулю 998 244 353.

Задача H. Супердевятка

Для начала, заметим, что любая «супердевятка» содержит ровно 12 игр. Действительно, всего есть $\frac{9 \cdot 8}{2} = 36$ различных пар участников. Каждая игра содержит 3 пары игроков. Каждая пара игроков должна встретиться ровно один раз. Таким образом, количество игр должно равняться $\frac{36}{3} = 12$.

Для решения задачи можно воспользоваться рекурсивным перебором. Во-первых, проверим что среди игр во входных данных никакая пара игроков не встречается более одного раза. Затем, будем рекурсивно перебирать различные наборы игр. Выберем игрока, у которого осталось как можно меньше других игроков, с которыми он еще не сыграл. Пусть этот игрок имеет номер a, а игроки, с которыми ему осталось сыграть, имеют номера b_1, b_2, $\dots b_k$. Заметим, что среди оставшихся игр обязательно будет игра вида (a, b_1, b_i), $2 \leqslant i \leqslant k$. Поэтому, для каждого $i \in [2, k]$, если b_1 и b_i еще не играли, можно добавить к множеству игр игру (a, b_1, b_i) и запуститься рекурсивно.

Если такой рекурсивной функцией мы дойдем до состояния, в котором все пары друг с другом сыграли, мы найдем ответ. С другой стороны, если ответ существует, мы таким рекурсивным перебором его гарантированно найдем.

Чтобы оценить время работы решения, нужно определить размер дерева рекурсивных вызовов. Его можно оценить как $7^3 \cdot 5^3 \cdot 3^3 \approx 10^6$. Тогда время работы не превышает $9 \cdot 10^6$.

Задача I. Шахматный конь

В этой задаче существует много различных решений. Мы приведем одно из конструктивных.

Для начала, выполним следующую последовательность операций, и будем запоминать, какие операции выполнились успешно:

1. $(1, -2)$
2. $(-1, -2)$
3. $(1, -2)$
4. $(-1, -2)$
5. $(2, -1)$
6. $(-2, -1)$

После их выполнения, конь гарантированно стоит на нижней горизонтали ($y = 0$). Если откатить успешно выполненные операции, мы узнаем y координату изначальной позиции коня. Теперь выполним следующую последовательность действий:

1. $(-2, 1)$
2. $(-2, 1)$
3. $(-2, 1)$
4. $(-1, 2)$

После ее выполнения, конь гарантированно стоит на левой вертикали ($x = 0$). Если откатить успешно выполненные операции до самого начала, мы узнаем x координату изначальной позиции коня. Всего было выполнено ровно 10 операций.

Задача J. Столетний дятел

Будем поддерживать текущую позицию корабля и направление, в котором он летит. А также, считать, сколько раз он уже повернул. За $O(n)$ можно найти ближайшее препятствие, которое корабль встретит. Если такого препятствия нет, то путешествие закончится без дополнительных поворотов, и можно вывести ответ. Иначе, переместим корабль до этого препятствия, повернем на 90 градусов, и увеличим ответ на 1.

Если в какой-то момент корабль прилетит в препятствие с той же стороны, с которой уже прилетал ранее, это означает, что он зациклился, и путешествие никогда не завершится. А если зацикливания не произойдет, то путешествие завершится после не более чем $4 \cdot n$ поворотов. Таким образом, решение работает за $O(n^2)$.

Также, можно написать решение, работающее за $O(n \cdot \log n)$. Для этого, нужно искать ближайшее препятствие в направлении движения за $O(\log n)$.

Задача K. Упавший сервер

Сформулируем условие, при котором место i может занять участник с номером j:

- не должно существовать записи l, r, m такой, что $l \leqslant i \leqslant r$ и $m > j$;
- не должно существовать записи L, R, M такой, что $L \leqslant i \leqslant R$ и $M < j$;
- не должно существовать записи l, r, m такой, что $i \notin [l; r]$ и $m = j$;
- не должно существовать записи L, R, M такой, что $i \notin [L; R]$ и $M = j$.

Построим двудольный граф. В первой доле n вершин — по одной для каждого места. Во второй доле n вершин — по одной для каждого участника. Проведём ребро между местом i и участником j, если данный участник может занять такое место. В полученном двудольном графе нужно найти лексикографически наименьшее совершенное паросочетание.

Для поиска максимального паросочетания можно воспользоваться алгоритмом Куна. Минимизировать ответ лексикографически можно жадно — попробуем поставить как можно меньшее число на первое место, потом на второе, и так далее. Соответственно, нужно уметь искать паросочетание при условии, что заданные пары (находящиеся на текущем префиксе перестановки) должны обязательно в паросочетание входить.

Поиск паросочетания работает за $O(n^3)$, поисков паросочетания нужно $O(n^2)$ (на каждую позицию слева направо попробовать поставить каждого участника), поэтому итоговая сложность алгоритма составляет $O(n^5)$.

Задача L. Развитие города

Построим «дерево поддеревьев» F. Корневым поддеревом будет исходное дерево, а при каждом подвешивании мы создадим вершину для нового поддерева, и подвесим её в дереве F к тому поддереву, к вершине которого мы подвешиваем поддерево в исходном дереве. Длину ребра в дереве F установим равной глубине вершины, к которой происходит подвешивание, относительно корня её поддерева, плюс один.

Как теперь вычислить глубину вершины v в дереве после всех подвешиваний? Найдём соответствующую ей вершину в F (поддерево исходного дерева). Сумма длин ребёр на пути от этой вершины до корня в F плюс глубина вершины v в исходном дереве относительно корня её поддерева — это и есть то, что нам нужно.

Как найти наименьшего общего предка вершин v и u после всех подвешиваний? Найдём соответствующие им вершины в F и найдём их наименьшего общего предка z (здесь z — это вершина F, соответствующая некоторому поддереву исходного дерева). Найдём, к каким вершинам поддерева z были подвешены те части дерева, в которых находились вершины v и u. Найдём наименьшего общего предка найденных двух вершин в исходном дереве. Его копия в поддереве z — это и есть то, что нам нужно.

Умея находить глубину и наименьшего общего предка, можно вычислить расстояние между вершинами v и u как сумму их глубин минус удвоенная глубина их наименьшего общего предка.

Задача M. Индекс примечательности

Для начала рассмотрим случаи $P = 2$ и $P = 5$. Как известно, делимость на 2 и 5 зависит только от последней цифры числа. Соответственно, для ответа на запрос нужно посчитать подстроки $[i, j]$ $(l \leqslant i \leqslant j \leqslant r)$, последняя цифра которых s_j делится на P. Пусть $p_1, p_2, \ldots, p_k$ $(l \leqslant p_1 \leqslant p_2 \leqslant \ldots \leqslant p_k \leqslant r)$ — это позиции в подстроке запроса, цифры на которых делятся на P. Тогда ответ на запрос равен $\sum\limits_{i=1}^{k}(p_i - l + 1) = (\sum\limits_{i=1}^{k} p_i) - k \cdot (l - 1)$. Чтобы найти число и сумму позиций с нужным свойством на подстроке, можно использовать два массива префиксных сумм.

Пусть теперь $P \neq 2$ и $P \neq 5$. Для каждого i $(1 \leqslant i \leqslant |T|)$ вычислим a_i — значение числа $T_i T_{i+1} \ldots T_{|T|}$ по модулю P, положим также $a_{|T|+1} = 0$. Значения a_i можно вычислить рекуррентно как $a_i = a_{i+1} + T_i \cdot 10^{|T|-i}$.

Тогда значение числа $T_i T_{i+1} \ldots T_j$ по модулю P равно $(a_i - a_{j+1})/10^{|T|-j}$. Поскольку 10 взаимнопросто с P (2 и 5 мы уже исключили из рассмотрения), подстрока $T_i T_{i+1} \ldots T_j$ делится на P тогда и только тогда, когда $a_i = a_{j+1}$.

Таким образом, ответ на запрос l, r равен числу пар i, j $(l \leqslant i < j \leqslant r + 1)$ таких, что $a_i = a_j$.

Для решения такой задачи можно воспользоваться так называемым *алгоритмом Мо*. Зафиксируем некоторое значение K и отсортируем все запросы по возрастанию $\lfloor l/K \rfloor$, а при равенстве — по возрастанию r. Будем обрабатывать запросы в этом порядке и переходить к следующему запросу расширением/сужением отрезка предыдущего запроса. Будем хранить мультимножество S значений a_i, находящихся в текущем отрезке. При расширении отрезка мы добавляем элемент в S, при сужении — удаляем. Также нужно параллельно с изменениями S поддерживать число пар равных элементов в S. В качестве S можно использовать либо хеш-таблицу, либо обычный массив (если заранее сжать значения a_i в интервал $[0; |T|]$). Можно показать, что при выборе $K = \sqrt{|T|}$ число операций добавления и удаления элемента равно $O((|T|+q)\cdot\sqrt{|T|})$, что укладывается в ограничение по времени.

Задача N. N-интересные числа

Заметим, что N-интересность числа определяется только его наибольшим простым делителем и их числом.

Заведём приоритетную очередь, в которую исходно положим все числа вида p^k, где p — простое, $p \leqslant 127$, $k \geqslant 1$ и $p^k \leqslant N$.

Далее n раз повторим следующую процедуру:

- Достанем наибольшее число x из приоритетной очереди. Если это n-е полученное число, выведем его и завершим исполнение.
- Для каждого простого делителя числа x сделаем следующее. Пусть этот делитель равен p. Если $p > 2$, найдём наибольшее простое число $q < p$ и положим в приоритетную очередь число $\frac{x}{p} \cdot q$.

Итого, начиная с чисел вида p^k, мы будем постепенно уменьшать их простые делители и таким образом сможем получить любое число, у которого k простых делителей и все они не превышают p (а все такие числа подходят под условие задачи).

Отметим, однако, что некоторые числа можно с помощью такой процедуры получить несколькими способами, и они окажутся в приоритетной очереди более одного раза. Бороться с этим можно несколькими путями:

- Проверять, что очередное число, которое мы достали из приоритетной очереди, не равно предыдущему.
- Использовать вместо приоритетной очереди структуру данных, хранящую только уникальные элементы (например, любое сбалансированное дерево поиска).
- Сделать так, чтобы процедура уменьшения чисел позволяла получить любое число уникальным способом. Например, при переборе простого p, которое мы заменяем на предыдущее простое $q < p$, достаточно пробовать заменить либо наибольший простой делитель (если его копий в числе x больше одной), либо второй по величине (если его копия в x ровно одна).

Задача O. Пожиратель кактусов

Переформулируем задачу. Выберем случайную перестановку вершин. Будем удалять вершины в порядке этой перестановки, каждый раз прибавляя к выделенной энергии размер ее компоненты связности в текущий момент. Несложно показать, что такой процесс эквивалентен исходному.

Пусть C_i — компонента связности вершины i перед ее удалением. В таком случае, нам необходимо посчитать $\mathbb{E}[\sum\limits_i C_i]$, которое в силу линейности равно $\sum\limits_i \mathbb{E}[C_i] = \sum\limits_i \sum\limits_j \mathbb{P}[j \in C_i]$. То есть фактически для каждой пары вершины необходимо найти вероятность того, что в момент удаления первой от нее был путь до второй.

Решим сначала задачу на дереве. Для того, что в момент удаления вершины i в ее компоненте связности была вершина j, необходимо, чтобы вершина i была удалена первой из всех вершин на пути между i и j, включая концы. Вероятность этого равна $\frac{1}{k}$, где k — количество вершин на этом пути. То есть ответ для дерева равен $\sum\limits_{i,j} \frac{1}{distance(i,j)+1}$, и его легко посчитать за квадратичное время.

Ответ для случая кактуса можно посчитать с помощью формулы включений-исключений. То есть необходимо:

- просуммировать вероятность того, что вершина будет удалена последней на каждом пути;
- вычесть вероятность того, что вершина будет удалена последней на каждом объединении двух путей;
- прибавить вероятность того, что вершина будет удалена последней на каждом объединении трех путей;
- ...

и так далее.

Формально, пусть P — множество простых путей из i в j. Тогда $\mathbb{P}[j \in C_i] = \sum\limits_{P' \subset P} (-1)^{|P'|} \frac{1}{|\bigcup P'|}$. Но в этой сумме элементов слишком много элементов, чтобы считать ее напрямую. Поэтому вместо этого будем с помощью динамического программирования считать сумму коэффициентов при вершинах с фиксированным размером объединения.

Кактус можно представить в виде дерева блоков-точек сочленения (https://neerc.ifmo.ru/wiki/index.php?title=Граф_блоков-точек_сочленения), причем каждый блок будет циклом. Подвесим его по очереди за каждую вершину. С помощью динамического программирования будем считать суммарный коэффициент при $\frac{1}{d}$ для всех d от корня поддерева для всех вершин в поддереве. Если вершина дерева блоков — обычная вершина, то ответ для нее — сумма ответов по всем детям, сдвинутая на 1. Если же вершина дерева блоков — это цикл, то необъодимо просуммировать ответы, по всем вершинам в цикле со сдвигом на количество взятых вершин из цикла. При этом для каждой вершины есть три варианта. Нужно или взять одну ветку цикла, или другую, или весь цикл. Причем, весь цикл нужно взять с коэффициентом -1.