Задача A. Стрельба из пушки

Автор задачи и разработчик: Даниил Голов

Давайте заметим, что прямой выстрел до цели, преодолевающий наикратчайшую дистанцию, и пролетающий над препятствием, идет по вектору, направленному из точки $(0, 0)$ в точку (d, w). Соответственно, если вектор с таким направлением и длиной k имеет X-координату меньше $2d$, то достать цель невозможно.

Если же X-координата такого вектора больше $2d$, то достаточно рассмотреть два случая:

- если $2w \leqslant h$, то прямой выстрел достигнет цели, и искомый угол равен $\tan^{-1}(\frac{w}{d})$;
- если же $2w > h$, то под таким углом снаряд перелетит дом, и необходимо использовать суперспособность NX5 с уходом пучка вниз, чтобы перебросить стену: в таком случае траектория полета пучка будет гипотенузой длины k прямоугольного треугольника с одним из катетов длины $2d$, то есть искомый угол будет составлять $\tan^{-1}\left(\frac{\sqrt{4k^2-4d^2}}{2d}\right)$.

Для полного решения достаточно просто проверить, какой из двух возможных вариантов подходит, и вывести любой подходящий. Во втором случае необходимо и достаточно, чтобы снаряд, запущенный под таким углом, не попал в препятствие. Все вычисления, кроме итогового вычисления угла, могут быть сделаны в целых числах.

Задача B. Иерархия цитадели

Автор задачи: Александр Гордеев, разработчик: Даниил Орешников

Переформулируем задачу: в дереве, в котором все листья расположены на одном и том же уровне, требуется отсортировать эти самые листья по возрастанию, меняя порядок детей у каких-либо вершин.

Заметим следующий факт: если листья в поддереве какой-либо вершины не образуют множество подряд идущих чисел, то отсортировать их не получится. И, наоборот, если поддерево каждой вершины содержит листья с подряд идущими номерами, то есть способ отсортировать их. Действительно, если в каком-то поддереве найдутся i и k, что существует j между ними ($i < j < k$) не из этого поддерева, то эта тройка листьев не сможет быть упорядочена: либо j будет стоять после k, либо до i.

Обратное утверждение следует из алгоритма решения: сделаем `dfs` (на самом деле в рамках данной задачи достаточно просто пройти по внутренним вершинам дерева в порядке убывания их номеров), и для каждой вершины посчитаем, чему равны минимальный и максимальный номера листьев в ее поддереве, проверив, что между этими значениями лежит ровно столько же чисел, сколько листьев в поддереве.

Когда эта информация посчитана для всех u_i — детей вершины v, отсортируем их в порядке возрастания номеров листьев в поддеревьях, и проверим, что для любых двух соседних детей выполняется $\texttt{min\_leaf}(u_i) = \texttt{max\_leaf}(u_{i-1}) + 1$. Только если все такие равенства выполняются, листья поддерева вершины v и поддеревьев всех ее потомков образуют отрезки подряд идущих значений, а подходящий способ сортировки — это перестановка u_i в полученном нами отсортированном порядке.

Для каждой вершины мы сортируем всех ее детей, что дает асимптотику времени работы $\mathcal{O}(n \log n + m)$.

Задача C. Защитное поле

Авторы задачи: Владислав Власов и Даниил Орешников, разработчик: Арсений Кириллов

Заметим следующий факт: если существует круг радиуса r, покрывающий половину точек, то существует и круг того же радиуса, покрывающий половину точек, на границе которого лежат хотя бы две из данных точек. Это можно доказать следующим образом: рассмотрим такой круг, будем двигать его в какую-либо сторону, пока он не коснется одной из точек. Затем будем вращать его относительно этой точки, пока он впервые не коснется второй. Поскольку мы рассмотрели первые моменты касания, то все точки, которые были внутри, остались внутри.

Сделаем двоичный поиск по ответу — с помощью него угадаем радиус искомого круга. Для того, чтобы проверить, существует ли окружность радиуса $r = \frac{l+r}{2}$, покрывающая хотя бы половину точек, переберем пару точек, на которые она будет опираться (выше доказано, что достаточно перебрать только окружности, опирающиеся хотя бы на пару точек из множества). Для каждой фиксированной пары точек существует не более двух способов опереть на них окружность, а для каждого из этих двух способов можно за время $\mathcal{O}(n)$ найти количество точек, попадающих внутрь соответствующей окружности. Достаточно просто найти ее центр, и для каждой точки определить, правда ли, что расстояние от нее до центра не превосходит r.

Таким образом, получаем решение, работающее за время $\mathcal{O}(n^3 \cdot \log(10^{15}))$, что при аккуратной реализации проходило все тесты.

Задача D. Симметричные карты

Автор задачи: Александр Гордеев, разработчик: Константин Бац

По условию, требовалось найти количество строчек A, B и C соответствующих длин таких, что все три строчки AB, AC, BC будут являются палиндромами. Для этого необходимо, чтобы символы в некоторых позициях строк совпадали. Разобьём все позиции на группы такие, что в каждой группе позиций символы должны совпадать. Тогда ответ на задачу равен 10^m, где m — максимальное количество таких групп, потому что каждой группе равных цифр можно независимо назначить любое значение.

Количество групп равных позиций можно найти при помощи системы непересекающихся множеств. Дадим позициям в строчках A, B, C общую нумерацию. Каждая позиция — одноэлементное множество в СНМ. Теперь по отдельности рассмотрим строчки AB, AC, BC и объединим множества, в которые входят позиции, равноудаленные от концов строк. Например, A_1 должно совпадать с $B_{|B|}$, поэтому объединим два соответствующих этим позициям множества в СНМ.

После всех объединений количество непересекающихся множеств и будет равно числу m. Для получения числа $10^m \bmod (10^9 + 7)$, которое требовалось найти по условию задачи, необходимо воспользоваться алгоритмом быстрого (логарифмического) возведения в степень.

Задача E. Эффективный двигатель

Автор задачи и разработчик: Мария Жогова

Докажем, что в конце будут активными только те карманные вселенные, которые являются полными квадратами. Посмотрим на произвольное число m и на его разложение на простые:

$$m = p_1^{a_1} \cdot p_2^{a_2} \cdot \ldots \cdot p_k^{a_k}.$$

Известно количество его натуральных делителей, оно равно в точности $(a_1+1)\cdot(a_2+1)\cdot\ldots\cdot(a_k+1)$. Действительно, каждое простое из разложения m входит в его делитель независимо от других со степенью от 0 до соответствующего a_i включительно.

Заметим, что такое произведение нечетно тогда и только тогда, когда все a_i четны. А это равносильно тому, что $\sqrt{m}$ — целое число, то есть m — полный квадрат. А если карманная вселенная номер m активна, это как раз и равносильно тому, что количество делителей m нечетно, ведь изначально все вселенные заморожены, а дальше меняют свое состояние перед каждым полетом, номер которого является делителем m.

Посчитать количество полных квадратов от 1 до n можно за время $\mathcal{O}(1)$, просто вычислив корень из n и округлив его вниз.

Задача F. Артефакты

Авторы задачи и разработчики: Владимир Новиков, Максим Дмитриев

У авторов есть два совершенно разных решения, в разборе будут оба.

Сразу обозначим за $mask$ битовую маска длины k, в которой i-й бит $mask$ равен 1, если и только если мы уже когда-то посещали артефакт с номером i.

Утверждение: оптимальный обход — это сумма ребер, взятых дважды, минус сумма ребер на диаметре нашего пути. Это следует просто из рассмотрения выбранного маршрута: его можно представить как непрерывный простой путь, от которого ответвляются поддеревья, на которых каждое ребро обходится в обе стороны (чтобы обойти все дерево и вернуться на «основной» маршрут).

Первое решение — Алгоритм Дейкстры. Действительно, давайте для каждой вершины создадим 2^k её копий. Пусть $\texttt{dp}[i][mask]$ — минимальный ответ на задачу, если мы закончили наш путь в вершине i, предварительно посетив $mask$ артефактов. Тогда мы можем запустить алгоритм Дейкстры из всех вершин i с $mask = 0$. Теперь осталось научиться обновлять нашу $mask$.

Пусть мы стоим в вершине i с артефактом $a[i]$. Тогда обозначим $new\_mask = mask$ `or` $2^{a[i]}$, то есть маска с добавленным артифактом $a[i]$. Попытаемся обновить ответ для $\texttt{dp}[i][new\_mask]$ через $\texttt{dp}[i][mask]$, после чего присвоим $mask = new\_mask$ (несложно понять, что нам надо выполнять такое присваивание в любом случае). Затем пройдемся по ребрам, исходящим из вершины i, и попробуем обновить ответ для $\texttt{dp}[j][new\_mask]$. Ответом на задачу будет минимум по всем $\texttt{dp}[i][2^k - 1]$. Если же мы не смогли посетить ни одну такую $mask$, то ответ на задачу равен -1. Итоговая асимптотика будет равна $\mathcal{O}(n \cdot 2^k \cdot \log(n \cdot 2^k))$. Подметим, что в этой задаче стоит написать Дейкстру через очередь, а не через `set`.

Второе решение — динамическое программирование по подмаскам и поддеревьям. Давайте заведем $\texttt{dp}[i][mask][flag]$ — минимальный ответ на задачу, если дерево подвешено за вершину i и мы собрали $mask$ артефактов с некоторым условием $flag$.

- Пусть $flag = 1$. Это означает, что нам надо два раза пройти по каждому ребру (т.е. полностью обойти дерево, даже по путям вверх).
- Пусть $flag = 2$. Это означает, что нам надо два раза пройти по некоторым ребрам (т.е. полностью обойти дерево), но при этом по какому-то пути вверх мы пройдем только один раз.
- Пусть $flag = 3$. Это оптимальный ответ для пункта i, если мы уже «вычли» диаметр.

Рассмотрим как мы пересчитываем данное `dp`. Пусть мы стоим в вершине u и переходим в вершину v. Перебор всех различных масок и подмасок занимает 3^k, все переходы мы делаем за $\mathcal{O}(1)$. Ответ нам надо обновлять из состояний $\texttt{dp}[i][2^k - 1][3]$. И того наша асимптотика времени работы $\mathcal{O}(n \cdot 3^k)$.

Задача G. Незваные гости

Автор задачи: Даниил Голов, разработчик: Константин Бац

Найдем сначала максимальное количество существ. Заметим, что количество уникальных посетителей в каждой группе может быть не больше, чем существ прилетело на Землю в каждой из групп. На самом деле, столько различных существ действительно могло побывать на Земле. Давайте все существа прилетят до того, как кто-то улетит. Тогда в момент, когда все прилетят, существ на Земле будет столько, сколько всего было посетителей в каждой из групп.

Найдем теперь минимально возможное количество различных посетителей. Заметим, что если в какой группе ни одно существо ни разу не посетило Землю, то ответ для такой группы — 0. Иначе ответ хотя бы один. При этом последовательность событий в логе можно выстроить в пары событий: «существо прилетело» – «существо улетело». При этом, если событий вида «существо прилетело» больше, чем событий «существо улетело», то различных существ на земле побывало не меньше, чем разность количества событий этих двух видов.

Таким образом, для каждой группы необходимо подсчитать количество событий «существо прилетело», пусть оно равно cnt_+, и количество событий «существо улетело», пусть оно равно cnt_-. Тогда, максимальное количество различных существ, побывавших на Земле, равно cnt_+. Если $cnt_+ > 0$, то минимально возможное количество различных посетителей равно $\max(cnt_+ - cnt_-, 1)$, иначе 0.

Время работы такого решения — $\mathcal{O}(n + m)$.

Задача H. Портальная пушка

Автор задачи и разработчик: Даниил Орешников

Сразу заметим, что третий запрос стандартно решается полиномиальным хэшированием. Если мы можем для каждой подстроки строки быстро узнавать ее хэш, то можем быстро сранивать подстроки на равенство. Осталось быстро обрабатывать запросы первых двух типов.

Научимся быстро обновлять хэши для подстрок. Для этого будем отдельно обрабатывать каждый возмжный символ от '`a`' до '`z`'. Заведем на каждый символ декартово дерево по явному ключу, которое будет хранить позиции вхождения этого символа. Помимо этого будем в каждой вершине ДД поддерживать сумму prime^i по всем позициям i в поддереве.

Чтобы посчитать хэш подстроки, будем итерироваться по всем символам и в каждом делать `split` соответствующего ДД, чтобы получить суммарный хэш позиций этого символа на отрезке. Хэш строки, соответственно, будет равен сумме $c \cdot \texttt{hash}(c, l, r)$ по всем возможным символам c. Единственное отличие, что хэш будет умноженным на prime^l, поэтому при сравнении хэшей надо умножить хэш самой левой подстроки на $\text{prime}^{|l_2 - l_1|}$.

Чтобы отвечать на запросы первых двух типов,

1. для запроса первого типа просто перенесем соответствующий индекс i из дерева символа c_1 в дерево символа c_2,
2. для запроса второго типа перенесем все элементы меньшего дерева в большее.

Поскольку мы всегда переливаем меньшее дерево в большее, каждый элемент будет перенесен не более $\log n$ раз, за исключением тех элементов, которые переносятся запросами первого типа. Таким образом, получим время работы $\mathcal{O}(26 \cdot (n + q) \cdot \log^2 n)$.

Задача I. Побег из космической тюрьмы

Автор задачи и разработчик: Даниил Орешников

Воспользуемся стандартной идеей разбиения перенстановки на циклы. Рассмотрим произвольный индекс i, а затем посмотрим на $p[i]$, $p[p[i]]$, и так далее. Рано или поздно этот ряд зациклится, образуя цикл, по которому элементы переставляются при применении данной перестановки.

Рассмотрим все элементы исходного массива и соответствующие им циклы. Заметим, что впервые каждый элемент окажется «на своем месте» (что позволит его затем использовать с помощью запоминающего устройства) через столько шагов, через сколько на цикле расположено равное ему число.

Таким образом, мы свели задачу к тому, чтобы для каждого числа найти ближайшее равное ему на его цикле. В частности, на цикле, на котором число встречается только один раз, его следующее появление будет только на шаге, равном длине цикла. Чтобы в общем случае для каждой позиции найти следующее появление равного числа, заведем `HashMap` (`unordered_set`), в котором будем хранить последнее вхождение каждого числа, и проитерируемся по удвоенному циклу, обновляя актуальную информацию.

Когда мы для каждого элемента нашли ближайшее вхождение равного ему в цикл, останется только взять максимум из всех полученных величин, что и будет ответом. Время работы этого решения равно $\mathcal{O}(n)$.

Задача J. Халат Рика

Авторы задачи: Константин Бац, Мария Жогова и Даниил Орешников, разработчик: Мария Жогова

Рассмотрим путь раствора с отдельной точки халата, на которую он был нанесён.

По условию, длина этого пути — расстояние от этой вершины до ближайшей дырки. Поскольку вес любого ребра в нашем графе равен единице, то для поиска кратчайшего расстояния до любой дырки в таком графе можно использовать поиск в ширину (`bfs`). Запустим множественный `bfs` из всех дырок в халате, чтобы узнать расстояние от них до всех капель. Если бы нельзя было добавить новую дырку в халате, то ответом был бы максимум из этих расстояний.

Теперь попробуем улучшить ответ, добавив новую дырку. Сделаем бинпоиск по ответу. Очевидно, что если можно достичь ответа $\leqslant d$, то можно и достичь ответа $\leqslant d + 1$. Осталось научиться проверять, что можно достичь ответа $\leqslant d$.

Для тех капель, для которых расстояние до исходных дырок не больше d, больше ничего делать не надо. Для оставшихся капель достаточно найти вершину на расстоянии не больше d до каждой из них, после чего сделать в ней дырку. Для этого сделаем `bfs` из каждой такой капли по отдельности и найдем для каждой оставшейся вершины графа максимальное из кратчайших расстояний до капель. Если нашлась вершина на расстоянии не больше d от каждой из капель, то в бинпоиске двигаем правую границу, иначе левую. Поскольку гарантируется, что капель не больше $s \leqslant 100$, время работы такого решения будет равно $\mathcal{O}(\log n \cdot (n+m) \cdot s)$, что проходит по времени.