Разбор задачи «Ральф и арифметика»

Пусть l_n — длина десятичной записи числа n.

Посчитаем количество чисел без второстепенных цифр длины $l = 1, 2, \dots l_n - 1$. Всего у нас k второстепенных цифр, значит всего у нас $10 - k$ разрешенных цифр. Также первая цифра не может быть нулем, значит вариантов первой цифры либо $f = 10 - k$, если 0 — второстепенная цифра, либо $f = 9 - k$ иначе. Тогда ответ для фиксированной длины l равен $f \cdot (10 - k)^{l-1}$.

Теперь посчитаем количество чисел без второстепенных цифр длины l_n. Все такие числа не превосходят n. Рассмотрим число n отдельно: прибавим 1 к ответу, если в записи числа n нет второстепенных цифр. Осталось рассмотреть числа длины l_n, строго меньшие n. У таких чисел есть несколько общих старших цифр с числом n (возможно, 0), затем идет цифра, меньшая соответствующей цифры числа n, а затем любые другие цифры. Переберем количество общих старших цифр с числом n: $p = 0 \dots l_n - 1$. Если среди первых p цифр числа n есть второстепенная, то таких чисел нет. Иначе нужно найти s — количество цифр, не являющихся второстепенными, строго меньшие p-й цифры числа n, и прибавить к ответу $s \cdot (10 - k)^{l_n - p - 1}$.

Разбор задачи «Битовый автомат»

Пусть a_{min} — это минимально возможный реальный урон, а a_{max} —максимально возможный реальный урон.

Заметим, что если $a = 0$, то оно минимально возможное, а если число $a = 2^n - 1$, то оно максимально возможное. Соответственно, для первого случая $a_{min} = a$, а для второго — $a_{max} = a$.

Если a не минимальное, то чтобы найти такое a_{min}, которое получается из a изменением одного бита, необходимо заменить самую старшую единичку в битовом представлении на ноль, поскольку это гарантирует максимальную разницу между a и a_{min}. Аналогично, чтобы получить максимальную разницу между a_{max} и a, необходимо заменить самый старший ноль битового представления a на единицу. При этом необходимо учитывать, что битовое представление числа a фиксированно и имеет длину n.

Пройдём по всем n битам числа a и посчитаем таким образом a_{min} и a_{max}.

Разбор задачи «Монетки»

Применим алгоритм, похожий на решето Эратосфена. Возьмем последовательность $a_i = i^2 + 1$: 2, 5, 10, 17, 26, 37, 50, 65, ...

Возьмем первое число — $2 = 1^2 + 1$. Посмотрим, на числа вида $(1 + 2 \cdot 0)^2 + 1$, $(1 + 2 \cdot 1)^2 + 1$, $(1 + 2 \cdot 2)^2 + 1$, ... Поделим все эти числа на 2 (на максимальную степень двойки, на которую они делятся), получим обновленную последовательность: 1, 5, 5, 17, 13, 37, 25, 65, ...

Теперь возьмем следующее число — $5 = 2^2 + 1$. Посмотрим, на числа вида $(2 + 5 \cdot 0)^2 + 1$, $(2 + 5 \cdot 1)^2 + 1$, $(2 + 5 \cdot 2)^2 + 1$, ... Поделим их все на 5 (тоже на максимальную степень 5, на которую они делятся): 1, 1, 5, 17, 13, 37, 1, 65, ...

Теперь возьмем третье число — 5, опять проделаем ту же самую операцию, получим: 1, 1, 1, 17, 13, 37, 1, 13, ...

Несложно доказать, что каждый очередной делитель (1, 5, 5, ...), который мы берем — простое число или 1. Таким образом, мы находим все простые делители чисел вида $n^2 + 1$.

Алгоритм работает за $O(n + n/2 + n/3 + ... + n/n)$, что равно $O(n \log(n))$.

Разбор задачи «Уничтожение дронов»

Для каждого дрона оценим через сколько он сможет добраться до Ральфа, если будет двигаться по оптимальному маршруту. За каждый ход дрон может менять обе свои координаты на 1. Таким образом дрону с в точке с координатами x_i и y_i понадобится $|x_i|$ ходов, чтобы оказаться на прямой $x = 0$, и $|y_i|$ ходов, чтобы оказаться на прямой $y = 0$. Соответственно в точке (0, 0), в худшем для Ральфа случае, он сможет оказаться через $\max(|x_i|, |y_i|)$ ходов. Примем эту величину за расстояние дронов до Ральфа.

Заметим, что выгоднее всего для Ральфа стрелять по дронам, расстояние до которых меньше. Отсортируем дронов по увеличению расстояния.

В каком случае Ральф не успеет уничтожить i-го дрона? В том случае, если расстояние до него будет меньше, чем число дронов с расстоянием не больше чем у него. Таким образом пройдемся по дронам в порядке увеличения расстояния и проверим, что $\max(|x_i|, |y_i|) >= i$ (нумерация дронов с 1). Если для всех дронов это выполнено, то можно вывести их в порядке увеличения расстояний, иначе решений не существует.

Разбор задачи «Конфета в лабиринте»

Заметим, что если можно пронести леденец длины k, то можно и всех меньших длин. Поэтому сделаем бинарный поиск по ответу. Что бы проверить, можно ли пронести леденец длины k, сделаем обход в глубину, где вершинами будут пары из самой верхней левой клетки леденца и направления, которому параллелен леденец. Из каждого такого состояния есть 6 переходов — два, в направлении сдвига на одну клетку вдоль леденца, и еще повороты на 90° вокруг одного из концов. Что бы проверить, что мы не заходим на стену, можно предподсчитать префиксные суммы по каждой строке и столбцу. Итого решение за $O(n \cdot m \cdot \log(\min(n, m)))$.

Разбор задачи «Женитьба»

Для решения задачи можно применить жадный алгоритм. Заметим, что если в исходной расстановке детей взять ближайших друг к другу мальчика и девочку и объединить их в пару, то ни у одного из них не может возникать соблазна, так как они являются наиболее симпатичными друг для друга. После того, как мы объединили этих детей в пару, можно заметить, что на оставшемся разбиении на пары они никак не могут сказаться, ведь, как мы уже выяснили, ни у одного из них не может возникать соблазн. Значит, про этих мальчика и девочку можно забыть и решать задачу независимо для всех остальных детей. Теперь среди этого множества детей выберем ближайших мальчика и девочку, и повторим весь процесс заново. После n таких операций объединения ближайших мальчика и девочки в пары, мы получим разбиение на пары, в котором не может возникать соблазнов.

Таким образом, мы получаем следующий алгоритм: нужно n раз взять пару из мальчика и девочки, расстояние между которыми минимально среди всех возможных пар, объединить их друг с другом и убрать из рассмотрения. Легко понять, что такой жадный алгоритм дает нам корректное разбиение на пары, а значит, ответа -1 в задаче быть не может.

Осталось лишь эффективно реализовать данный алгоритм. Для этого можно сделать следующее замечание: если расположить всех детей в порядке возрастания координаты на прямой, то парой из мальчика и девочки, расстояние между которыми минимально, обязательно будет пара, в которой эти мальчик и девочка стоят подряд. Таким образом, достаточно поддерживать структуру, в которой можно хранить упорядоченных по возрастанию координаты детей, а также удалять их из этой структуры, когда соответствующий ребенок объединяется в пару и исключается из рассмотрения. В качестве такой структуры пойдет, например, `std::set` в языке C++ или `TreeSet` в Java. В такой же структуре можно поддерживать отрезки между соседними детьми разного пола, храня их в порядке возрастания длины. Тогда чтобы выбрать очередную пару, надо лишь взять минимальный по длине отрезок из этой структуры, а при удалении детей из рассмотрения нужно удалять старые отрезки, которые они образовывали, и добавить новые. Легко заметить, что таких отрезков всегда будет константное число.

Таким образом, если запросы ко всем структурам работают за время $\mathcal{O}(\log n)$, то асимптотика времени работы решения составит $\mathcal{O}(n \log n)$.

Разбор задачи «Игра с деревом»

Рассмотрим строки в порядке от вершины к корню. Тогда количество различных подслов равно количеству различных префиксов таких слов.

Научимся считать количество различных префиксов в наборе слов. Отсортируем все строки. Пусть lcp_i — количество первых букв, которые совпадают у i-й и $i+1$-й строки. Тогда количество различных подстрок равно $\sum_{i=1}^{n} len_i - \sum_{i=1}^{n-1} lcp_i$.

Чтобы пересчитывать количество различных строк, будем добавлять строку в нужное место в отсортированный массив и пересчитывать ответ. Пусть новая строка добавилась в позицию i. Тогда количество новых различных подстрок равно $len_i - lcp(i, i+1) - lcp(i-1, i) + lcp(i-1, i+1)$.

Чтобы быстро добавлять строку, будем поддерживать упорядоченное множество (set) по лексикографическому порядку слов. Тогда операция вставки будет работать за $O(comp \cdot \log n)$, где $comp$ — время сравнения двух строк, а пересчет за $O(lcp)$ — время нахождения lcp.

Насчитаем от каждой вершины $up_{v,i}$ — предка v на расстоянии 2^i, и $h_{v,i}$ — полиномиальный хеш строки длины 2^i, начинающейся в вершине v:

$up_{v,i} = up_{up_{v,i-1},i-1}$

$h_{v,i} = h_{v,i-1} \cdot p^{2^{i-1}} + h_{up_{v,i-1},i-1}$.

С помощью этого можно сравнивать строки и искать lcp за $\mathcal{O}(\log n)$, аналогично двоичным подъемам.

Время работы: $\mathcal{O}(n \cdot \log^2(n))$.

Разбор задачи «Пароли»

Заметим, что условие задачи можно переформулировать следующим образом. Посчитайте количество способов разбить s на три строки a, b и c, так что $a \neq b$, $b \neq c$ и $a + b \neq b + c$.

Количество способов разбить s на три отрезка — $C^2_{|s|-1}$.

Вычтем количество способов, в которых $a = b$ или $b = c$. Чтобы посчитать количество разбиений, в которых $a = b$, переберем четную длину $l \cdot 2$ строки $a + b$, и проверим, что $s[1 \dots l] = s[l + 1 \dots l \cdot 2]$ (для этого можно, например, использовать z-функцию). Количество разбиений, в которых $b = c$, считается аналогично. Обратите внимание, что возможен случай, когда $a = b = c$. Его нужно учесть, и в таком случае, прибать 1 к ответу.

Осталось вычесть количество разбиений, в которых $a \neq b$, $b \neq c$ и $a + b = b + c$. Для этого, переберем длину l строки $a+b$. Проверим, что $s[1 \dots l] = s[|s|-l+1 \dots |s|]$, $s[1 \dots |s|-l] \neq s[|s|-l+1 \dots l]$ и $s[|s| - l + 1 \dots l] \neq s[l + 1 \dots |s|]$. Для этого тоже можно использовать z-функцию.

Разбор задачи «Случайное дерево»

Подвесим дерево за любую вершину.

Тогда количество подмножеств, в которых эта вершина не присутствует, это $\sum (2^{sz_{to}} - 1)$ (по всем детям to вершины v) $+ 2^{n-sz_v}$, где sz_v — размер поддерева v.

Таким образом, ответ на задачу это $2^n \cdot n - \sum (2^{sz_v} - 1) + \sum 2^{n-sz_v} - 2^n + 1$.

Также можно показать, что математическое ожидание глубины вершины в случайном дереве это $\log n$.

Тогда можно подняться за $O(depth)$ и пересчитать размеры поддеревьев, и вместе с ними обновить ответ.

Итоговое время работы: $\mathcal{O}(n \log n)$.

Разбор задачи «Праздничные вычисления по сахарному модулю»

Для начала узнаем, какие биты являются единичными в $x \oplus y$. Тогда мы сможем представить результат в виде суммы 2^i, по всем битам i, равным 1 в $x \oplus y$.

Не умоляя общности предположим, что $x > y$. Затем получим $2^0 = 1$, деля число y целочисленно на 2, пока не получим 1. На это у нас уйдет не более $O(\log(y))$ операций. Теперь достаточно с помощью операции умножить на 2, получить из 2^0 нужные степени двойки и сложить их. Нужных степеней двоек будет порядка $O(\log(x))$.

Для удобства обращения к памяти можно было сначала получить все степени двойки, а уже потом выполнять сложение.

Всего нужно будет хранить порядка $O(\log x)$ переменных.