Разбор задачи «Re: Fwd: Про чайник»

Изначальны было $n \cdot a$ литров воды Если воды хватит на промывку чайника, то после того, как чайник будет промыт, останется $n \cdot a - x$ литров воды. После равномерного распределения воды, каждому из m людей достанется по $\lfloor \frac{n \cdot a - x}{m} \rfloor$ литров. Значит, если $n \cdot a - x \geqslant 0$, то ответ будет $\lfloor \frac{n \cdot a - x}{m} \rfloor$, а иначе воды не хватит даже на промывку чайника и ответ на задачу – 0 литров.

Разбор задачи «Проверка олимпиады»

У этой задачи много разных решений, приведём конструктивное.

Переформулируем задачу так: необходимо построить таблицу C_{ij} размера $t \times n$, чтобы в каждой строке стояли числа от 0 до m, и чтобы каждое положительное число в одной строке не повторялось. Кроме того, в каждом столбце различных положительных чисел должно быть хотя бы k.

Временно забудем про числа, и поставим вместо положительных чисел единицы. Построим эту таблицу следующим образом: в первой строке закрасим клетки $1 \dots m$, во второй – $m+1 \dots 2m$, в третьей – $2m+1 \dots 3m$, и т.д. Как только мы дойдем до конца строки, перескочим на начало (т.е. мы красим вычеты по модулю n). Будем повторять эту процедуру столько раз, пока в каждом столбце не станет хотя бы k покрашенных клеток. Стоит отметить, что на каждом шаге алгоритма количества покрашенных клеток в различных столбцах отличаются не более, чем на 1. Отсюда следует, что меньшим числом строк обойтись нельзя.

Покажем, как расставить числа в строках. Для начала будем действовать так: будем расставлять числа, начиная с 1, в том порядке, в каком мы красили клетки в строке. К сожалению, это решение неверное, так как мы никак не учитываем то, что в каждом столбце должны стоять различные числа. Например, на тесте $n = 4, k = m = 2$ это решение выведет $\begin{matrix} 1 & 2 & 0 & 0 \\ 0 & 0 & 1 & 2 \\ 1 & 2 & 0 & 0 \\ 0 & 0 & 1 & 2 \end{matrix}$. Чтобы исправить это, рассмотрим первый момент, когда мы снова начали красить клетки от начала строки. Теперь вместо чисел $1, 2, \dots, m$ будем расставлять числа $2, 3, \dots, m, 1$, пока снова не вернемся в начало. Далее аналогично: каждый раз, когда мы стартуем из первой клетки, будем циклически сдвигать числа, которые мы вставляем. Докажем, что в итоге в каждом столбце будут стоять различные числа. Рассмотрим столбец j, посмотрим на числа, стоящие в нем, пока мы не вернулись в начало. Пусть первое число в этом столбце – x. Рассмотрим следующее за ним число, пусть это $x + a \mod m$. Нетрудно заметить, что в этом столбце также будут числа $x + 2a \mod m$, $x + 3a \mod m$ и т.д. Тогда это же самое множество вычетов – это арифметическая прогрессия с шагом $d \mid m$, где $d = gcd(a, m)$. Посмотрим, какие числа окажутся в столбце j, после того, как мы прошли через начало. Они будут прогрессией с тем же шагом d и начальным числом $x + 1$. Следовательно, она не будет пересекаться со старой (кроме случая, когда мы каждое число посетили, тогда в столбце уже m клеток покрашено, и мы остановимся). Следовательно, в каждом столбце все числа различны, т.е. их k или $k + 1$, а значит матрица C удовлетворяет условию задачи.

Разбор задачи «Дед и мопед»

Давайте для каждого города i найдём p_i — номер ближайшего города с заправкой, а также d_i – наименьшее расстояние до такого города (будем считать, что в начальном городе также расположена заправка). Сделаем это с помощью алгоритма Дейкстры. Для этого достаточно для всех городов с заправками сделать $d_i = 0, p_i = i$, а потом запустить алгоритм от всех заправок одновременно. Теперь допустим, что для какого-то ребра j, $d_{u_j} + d_{v_j} + c_j \leqslant s$, $p_{u_j} \neq p_{w_j}$. Это значит, что из вершины p_{u_j} можно добраться до вершины p_{v_j} и наполнить там бак (и наоборот). Давайте тогда просто добавим в граф ребро с $u = p_{u_j}, v = p_{v_j}, c = 0$. Теперь докажем, что если какая-то вершина с заправкой достижима из какой-то другой вершины с заправкой, то из неё существует путь стоимости 0 в получившемся графе.

Обозначим за $l(u, v)$ наименьшее расстояние между вершиными u и v. Допустим, утверждение неверно. Тогда найдутся вершины a и b, такие что в них есть заправки, и $l(a, b) \leqslant s$, при этом между ними нет пути длины 0 в получившемся графе. Пусть теперь кратчайший путь из a в b состоит из вершин $(u_1, u_2, ..., u_q)$, $u_1 = a$, $u_q = b$. Заметим, что

для любой вершины u_i, $l(u_i, p_{u_i}) \leqslant l(u_i, a)$, $l(u_i, p_{u_i}) \leqslant l(u_i, b)$. Теперь рассмотрим две соседние вершины пути u_i и u_{i+1}. Пусть они соединены ребром веса w. Мы знаем, что $l(u_i, p_{u_i}) + w + l(u_{i+1}, p_{u_{i+1}}) \leqslant l(u_i, a) + w + l(u_{i+1}, b) \leqslant s$, поэтому вершины p_{u_i} и $p_{u_{i+1}}$ соединены ребром веса 0. Но при этом $u_1 = a \Rightarrow p_{u_1} = a, u_q = b \Rightarrow p_{u_q} = b$, поэтому между вершинами a и b будет путь длины 0 в получившемся графе. Противоречие.

Поэтому нам осталось запустить алгоритм Дейкстры на графе с дополнительными рёбрами, и найти вершины на расстоянии не больше s от стартовой.

Разбор задачи «Задача для разминки рук»

Для решения задачи будем использовать центроидную декомпозицию дерева. Для каждого центроида будем рассматривать пути, которые проходят через него (внутри его компоненты) и лексикографически меньше строки s.

Каждый путь через центроид будет имеет какой-то (возможно нулевой) общий префикс со строкой s, а дальше происходит различие. Все пути через центроид делятся на 2 типа — те, у которых общий префикс с s закончился раньше центроида, и те, у которых общий префикс с s прошёл через центроид.

Научимся считать число интересующих нас путей первого типа. Для этого запустимся dfs-ом из центроида. Теперь у нас есть запрос добавить символ в начало строки (проход в ребёнка в dfs), удалить символ из начала строки (возвращение в отца в dfs) и узнать наибольший общий префикс со строкой s. Если развернуть обе строки, то теперь задача сводится к тому, чтобы добавить символ в конец строки, удалить символ из конца строки и узнать наибольший общий суффикс со развёрнутой строкой s.

Такую задачу можно решить при помощи суффиксного автомата. Построим суффиксный автомат для развёрнутой строки s. Далее, аналогично автомату Ахо-Корасика, для каждой вершины A и каждого символа c проведём переход из A по c в вершину автомата B такую, что максимальная строка, ведущая в B, является суффиксом максимальной строки, ведущей в A с приписанным символом c. Делать это будем ленивой динамикой, аналогичной построению автомата Ахо-Корасик — когда надо для вершины построить переходы, то для начала сделаем это для суффиксной ссылки из вершины, а дальше если из вершины нет перехода по символу c, то направляем его туда, куда ведёт переход по символу c из суффиксной ссылки. Так же для каждой вершины посчитаем первую достижимую по суффиксным ссылкам вершину, являющуюся терминальной (то есть совпадающей с некоторым суффиксом строки).

Теперь, чтобы научиться отвечать на наши запросы, будем для g хранить вершину A автомата такую, что в неё ведёт наибольшая по длине строка, являющаяся суффиксом g, и число l — длину этой строки. Когда добавляется символ c, то делается переход по c из A и это становится новой A, а l становится равно минимуму из $l+1$ и длины наибольшей строки, ведущей в новую A. Когда надо узнать наибольший суффикс g и развёрнутой s, то берётся ближайшая достижимая по суффссылкам терминальная вершина автомата из A и ответ равен минимуму из длины наибольшей строки, ведущей в эту вершину и числа l. Все операции выполняются за $O(1)$.

Теперь, для каждой первой половины пути до центроида мы знаем наибольший общий префикс со строкой s. Если длина префикса меньше длины пути, то смотрим на следующий символ пути после префикса. Если он меньше следующего символа строки, то любое продолжение лексикографически меньше s, иначе любое продолжение лексикографически больше s. Так мы научились считать число путей, у которых префикс с s заканчивается до центроида.

Теперь научимся считать число путей, у которых префикс с s заканчивается после центроида. У таких путей часть пути до центроида совпадает с префиксом s и все такие пути мы уже научились считать. Посмотрим на любой из них. Пусть у него длина l. Тогда продолжение пути после центроида (начиная с него) должно быть лексикографически меньше суффикса s, который начинается в l-й позиции.

Построим суффиксный массив для строки s. Теперь для каждой вершины v компоненты центроида мы найдём cnt_v — количество суффиксов строки s, лексикографически не меньших строки, соответствующей пути из центроида в v (обозначим s_v). Тогда если у суффикса, начинающегося в l-й позиции, позиция в суффиксном массиве не меньше cnt_v, то любой путь, которому соответствует

строка $s_{[1..l-1]}s_v$, нам подходит. Если же номер суффикса $s_{[l;m]}$ в суффиксном массиве меньше cnt_v, то такое продолжение не подходит. Тогда для каждой первой части пути до центроида, совпадающей с некоторым l-м префиксом s, найдём позицию в суффмассе $(l+1)$-го суффикса — обозначим за P мультимножество всех таких значений. Теперь для каждой второй половины пути v надо посчитать количество значений в P, не меньших cnt_v. Так как такие запросы приходят в оффлайне, то чтобы на них ответить за линейное время, достаточно отсортировать все запросы и позиции, что можно сделать для всех центроидов одновременно сортировкой подсчётом суммарно за $O(n \log n)$.

Осталось для каждой вершины v найти cnt_v — количество суффиксов строки s, лексикографически не меньших пути. Для этого построим суффиксный автомат строки s, в котором для каждого состояния найдём число достижимых терминальных вершин (то есть число суффиксов, у которых префикс совпадает с любой строкой, ведущей в вершину). Чтобы у каждой вершины найти cnt_v, запустимся dfs-ом из центроида. Для пути от центроида до текущей вершины будем хранить вершину A суффиксного автомата, соответствующую строке пути, и число x — количество суффиксов s, лексикографически не меньших текущей строки пути. При добавлении символа (спуску в ребёнка в dfs) из A совершается переход по этому символу. При этом для всех переходов по меньшим символам к x прибавляется число достижимых терминальных вершин по этим переходам.

Так при обработке центроида мы делаем O(размер компоненты центрода) действий, и в итоге получаем решение за $O(n \log n)$.

Данное решение может достаточно долго работать на дереве со случайной нумерацией вершин, одним из вариантов ускорения является перенумерация вершин в порядке hld-reorder (то есть у вершины поддеревья сортируются в порядке убывания размера поддерева).

Вообще говоря, авторское решение вместо сортировки подсчётом использует `std::sort` и потому имеет временную сложность $O(n \log^2 n)$, но из-за большой константы других компонент решения работает не хуже.

Также подсчёт путей можно реализовать немного другим образом. Мы можем считать для каждого центроида количество путей, проходящих через центроид, начало которых находится в компоненте, и часть пути до центроида совпадает с каким-то префиксом s. При этом конец может быть где угодно в дереве. Тогда для каждого центроида нас интересуют только такие первые части путей, строки которых совпадают с префиксом s. Проверку на совпадение можно реализовать обычным хешированием с модулем M и основанием p. При этом, чтобы хеши работали быстрее, можно заметить две вещи: во-первых, при использовании хешей мы только дописываем в конец некоторый символ, то есть домножаем хеш максимум на p — так как алфавит размера 3, эту величину можно взять равной 4. Таким образом, мы можем выбрать очень большое M, порядка 10^{18}. Таким образом медленное взятие по модулю M можно заменить на два вычитания, так как $p = 4$. Подсчёт количества продолжений путей делается аналогично предыдущему решению, но отдельно надо посчитать число продолжений, выходящих за пределы центроидной компоненты.

Разбор задачи «Проблема останова»

Пусть программа находится в процессе выполнения. Тогда пара из строки кода, на которой сейчас находится выполнение, и битовой маски, где единичные биты соответствуют единичным регистрам, а нулевые — нулевым, полностью описывают состояние выполнения.

Если мы в процессе выполнения оказались в состоянии, где уже были, то есть на той же строке кода и с такими же значениями регистров, это значит, что дальше мы повторим уже сделанные действия, снова попадем в это состояние, и таким образом окажемся в бесконечном цикле.

Представим себе граф, где вершинами являются такие пары. Ребро ведет из одной пары в другую, если выполнив строку кода, соответствующую первой паре, имея в регистрах из битовой маски первой пары, вторая пара будет описывать получившееся состояние выполнения.

В таком графе из каждой вершины, кроме тех, которых соответствуют последней строке программы (с `end`), исходит ровно одно ребро, а из соответствующих `end` — ни одного.

Если из какой-то из стартовых вершин (то есть соответствующих строке `begin`) в таком графе достижим цикл, это значит, что есть вход, на котором программа не останавливается.

Можно перебрать все возможные входные данные, просимулировать процесс и проверить, не окажемся ли мы в состоянии, в котором уже были в процессе выполнения для этих данных. Это

$O(4^k n)$

А можно сохранять состояния, которые мы посетили, выполняя программу для всех предыдущих входных данных. Если в процессе выполнения мы оказались в состоянии, в котором уже были ранее для других входных данных (и при этом еще не нашли бесконечный цикл), то для этих мы тоже его не найдем, и можно прервать выполнение для этих данных и перейти к следующим. Если же мы оказались в состоянии, в котором были, выполняя программу для текущих входных данных, то мы вошли в бесконечный цикл. Таким образом, каждую вершину графа мы посетим не более одного раза, и время работы составит $O(2^k n)$

Из этой задачи следует, что пока не изобрели бесконечную память для компьютеров, на практике проблема останова разрешима, пусть и за очень долгое время!

Разбор задачи «Футболки на олимпиаду»

Обозначим ответ на задачу за D, а общую вместимость площадок за $k := \sum_{i=1}^{m} a_i$

Сделаем бинпоиск по ответу на вопрос «Есть ли распределение, для которой максимальное расстояние не больше d?». Мы можем это сделать, потому что ответ на этот вопрос это «нет» для всех чисел, меньших D. При этом, для всех чисел, больших или равных D, ответ «да». Заметим, что бинпоиском можно перебирать не все расстояния, а только расстояния между какой-то парой «участник — площадка», ведь таких пар всего $O(nk)$

Теперь нам надо проверить, что есть перестановка, удовлетворяющая условию на то, что у каждого участника найдется площадка на расстоянии не больше, чем d.

Построим двудольный граф, в левой доли которого будут участники, а в правом — места на площадках. Тогда в левой доли будет n вершин, а в правой — k. Соединим ребром участника и место, если расстояние от участника до площадки не больше, чем d.

Теперь в этом графе нам надо каждой вершине левой доли найти пару из правой доли. Эта задача решается алгоритмом Куна, как задача о максимальном паросочетании. Так же эта задача может быть решена потоками, однако немного оптимизированного алгоритма Куна достаточно, чтобы набрать полный балл по задаче.

Итоговая асимптотика — $O(n^2 k \log(nk))$

Разбор задачи «Тир»

Первое что нужно сделать, рассмотреть задачу немного с другой стороны. Давайте посмотрим на множество клеток поля, в которых когда-либо находились перегородки. У каждой из них есть по 4 стороны, причём если конкретная пуля вышла из данной клетки из данной стороны, то она будет идти по прямой, пока не дойдёт до какой-то другой клетки, где когда-либо находилась перегородка, или же вылетит за любую из границ поля.

Тогда можно построить следующий граф: для каждой клетки, в которой когда-либо была перегородка заведём по 4 вершины, по одной на каждую сторону и ещё по одной вершине для каждой клетки соседней снаружи для крайних клеток. Изначально для каждой вершины порождённой клеткой, соединим её с ближайшей вершиной в которую попала бы пуля, если бы вылетела из данной клетки по этому направлению. Так же соединим внутри клетки вершины отвечающие за верх-низ и за право-лево. В таком графе добавление/удаление перегородки будет выглядеть как изменение ровно 2 рёбер внутри 4 вершин отвечающих за стороны, т.к. по сути, эти рёбра - это перенаправление пули, которая прилетает в перегородку. В такой постановке ответ - это количество компонент связности в которых есть одна вершина отвечающая за клетку соседнюю снаружи для вернего ряда и одна вершина отвечающая за клетку соседнюю снаружи для нижнего рядаю.

В итоге у нас есть множество вершин и множество рёбер, запросы: добавить ребро, удалить ребро, посчитать количество компонент связности, в которых есть 2 вершины из интересных множеств. Можно заметить, что граф, с которым мы работаем, состоит только из путей и циклов (так как степень любой вершины не превышает 2). Такие пути и циклы можно хранить в декартовом дереве, храня заодно информацию о том, есть ли интересные вершины в декартовом дереве.

Когда происходит удаление рёбер, то пути или циклы распадаются на части, а соответственно разделяются декартовы деревья. Когда происходит добавление рёбер, то если концы этого ребра

раньше находились в разных декартовых деревьях, то эти декартовы деревья можно развернуть так, чтобы начало ребра было последним в своём декартовом дереве, а конце ребра был первым в своём декартовом дереве. Дальше 2 декартовых дерева можно объединить. Если же оба конца уже находились в одном декартовом дереве, то теперь компонента связности становится циклом.

При изменении рёбер будут меняться только пути, содержащие эти рёбра, соответственно только декартовы деревья, хранящие эти пути. Поэтому каждое изменение можно сделать за $O(\log(n+m+q))$ и итоговая асимптотика: $O((n+m+q)\cdot\log(n+m+q))$

Разбор задачи «Украшение дома»

Каждый столбец, который надо покрыть (ненулевой) дает 2 ограничения на размеры. Первое ограничение — высота столбца. Второе ограничение это (количество столбцов слева больше или равных по высоте) + (количество столбцов справа больше или равных по высоте) + 1. Итоговое ограничение — суммарное расстояние до ближайшего меньшего слева и ближайшего меньшего справа.

Поэтому прямоугольник, которым мы будем покрывать, должен при какой-то ориентации быть меньше чем ограничения по высоте и ширине. Тогда большее из этих ограничений будет ограничением на длину большей стороны итогового прямоугольника, а меньшее из них — на длину меньшей стороны прямоугольника. Объединив все такие ограничения мы сможем найти итоговые ограничения на прямоугольник максимальной площади, которым можно покрыть стену. При этом стену обязательно получится покрыть, потому что таким прямоугольником можно будет покрыть каждый столбец стены.

Чтобы найти ближайший столбец слева или справа, имеющий меньшую высоту, можно поддерживать последовательность возрастающих столбцов в стеке или сете, или же это можно сделать спуском по дереву отрезков.

Разбор задачи «Доставка посылок»

Обозначим за W ограничение на w_i. Для каждого числа от 1 до W выпишем все его делители. Чтобы это сделать, переберём делитель d и для всех чисел вида d, $2d$, ... , $\lfloor\frac{W}{d}\rfloor d$ запишем d в список делителей. Этот предподсчёт будет работать за $O(W\cdot\log(W))$.

Теперь для каждого числа d выпишем все рёбра, у которых w_i кратно d. Это можно сделать, перебрав для каждого ребра все предподсчитанные делители w_i и записав ребро в соответствующие списки рёбер.

Будем перебирать число d и искать количество путей, по которым можно провезти посылку веса d. Для этого будем добавлять в пустой граф только рёбра из полученного ранее списка рёбер с w_i, кратными d. Далее будем находить размеры компонент связности в получившемся графе, запуская dfs только из вершин, у которых было добавлено хотя бы одно ребро. Если размер очередной компоненты связности равен sz, то к количеству путей нужно прибавить $sz(sz-1)$. После того, как нашли размеры всех компонент связности, нужно удалить все добавленные ребра из графа. Т.к. каждое ребро будет добавлено в граф и посещено dfs столько раз, сколько делителей у w_i, то эта часть решения работает за $O(n\cdot maxDiv)$, где $maxDiv$ — максимальное число делителей у числа (на практике $maxDiv$ имеет порядок $O(W^{1/3})$).

Пусть $cnt[d]$ - количество путей, по которым можно провезти посылку веса d. Нужно узнать величину $ans[d]$ - количество путей, НОД w_i на которых равен d. Можно вычислить её следующим образом: $ans[d] = cnt[d] - ans[2d] - ans[3d] - ... - ans[\lfloor\frac{W}{d}\rfloor d]$ (т.к. $cnt[d]$ - это суммарное количество путей, НОД на которых кратен d, т.е. равен одному из чисел d, $2d$, ... $\lfloor\frac{W}{d}\rfloor d$). Вычисление $ans[d]$ для всех d от 1 до W работает за $O(W\cdot\log(W))$. Для получения ответа на задачу нужно просуммировать числа вида $d\cdot ans[d]$.

Итоговая асимптотика решения: $O(W\cdot\log(W)+n\cdot maxDiv)$.

Разбор задачи «Тараканы общежития»

Предположим, что ответ для входных данных есть. Попытаемся его построить и если у нас не получается, то считаем, что ответа нет.

Для начала заметим, что если на прямой хотя бы $k+1$ точка, то по принципу Дирихле хотя бы две из них находятся на одной прямой в ответе, а значит текущая прямая должна быть в ответе и мы можем удалить эти точки, а затем уменьшить k на 1.

Тогда если точек больше, чем k^2, то возьмём произвольные k^2+1 точку. Если среди прямых, проходящих хотя-бы через 2 из них нет прямой, содержащей $k+1$ точку, то их нельзя покрыть k прямыми. Иначе берём прямую в ответ и повторяем то же уже для $k-1$ прямой Когда процесс останавливается мы получаем, что прямых не более, чем k^2.

Далее решение представляет из себя полный перебор, с возможными отсечениями. В зависимости от реализации решение может набирать различное число баллов.

Полное решение состоит из того, что все прямые рассматриваются в порядке убывания веса — число точек, которые находятся на рассматриваемой прямой. Также важное отсечение, что если вес текущей прямой умноженный на число оставшихся прямых в ответе меньше, чем число оставшихся точек, то можно уже не рассматривать эту ветку перебора. Практика показывает, что можно не пересчитывать вес (что позволяет не делать много лишних действий).

Почему данное решение быстро работает? Заметим, если у нас есть n точек, то есть прямая, которая содержит хотя бы $\frac{n}{k}$ точек. Часто таких прямых немного, что и позволяет воспользоваться этим.